Truthful Approximation Schemes for Single-Parameter Agents

Peerapong Dhangwatnotai[*] Shahar Dobzinski[†] Shaddin Dughmi[‡] Tim Roughgarden[§]

Abstract

We present the first monotone randomized polynomial-time approximation scheme (PTAS) for minimizing the makespan of parallel related machines ($Q||C_{\max}$), the paradigmatic problem in single-parameter algorithmic mechanism design. This result immediately gives a polynomial-time, truthful (in expectation) mechanism whose approximation guarantee attains the best-possible one for all polynomial-time algorithms (assuming $P \neq NP$). Our algorithmic techniques are flexible and also yield, among other results, a monotone deterministic quasi-PTAS for $Q||C_{\max}$ and a monotone randomized PTAS for max-min scheduling on related machines.

1 Introduction

Algorithmic mechanism design studies resource allocation problems where the underlying data (such as the value of a good or the cost of performing a task) is *a priori unknown* to the algorithm designer, and must be implicitly or explicitly elicited from self-interested agents (e.g., via a bid). There is a complex interaction between the way an algorithm employs this information and the behavior of the participants—for example, in a "first-price" auction (where winners pay their bids), bidders will shade their bids below their maximum willingness to pay, while in a "second-price" auction participants are incentivized to bid their true value for a good. Algorithmic mechanism design has applications in the design of auctions, contracts, pricing schemes, and so on (see e.g. [17]).

An important research agenda, suggested roughly ten years ago [18], is to rigorously understand what can and cannot be efficiently computed when the problem data is held by selfish agents, thereby reconciling strategic concerns with the computational requirements customary in computer science. In particular, one of the deepest open questions in algorithmic game theory is, roughly: *is "incentive-compatible" efficient computation fundamentally less powerful than "classical" efficient computation?*

By "incentive-compatible", we mean the following. Suppose each agent i holds a private cost function t_i and we want to optimize some objective function involving the t_i's over a set of feasible outcomes Ω. In our primary example, Ω is all schedules of n jobs with known sizes $p_1, \ldots, p_n$ on m parallel related machines (the agents); t_i is a function of the form $x_i(\omega)/s_i$, where s_i denotes the (private) speed of agent i's machine and $x_i(\omega)$ denotes the work (sum of job sizes) assigned to i in the schedule ω; and our objective is to minimize the *makespan* $\max_i t_i(\omega)$ of the machines. Every vector t of private data induces an instance of an optimization problem Π, which in our example is the strongly NP-hard problem $Q||C_{\max}$. An algorithm $\mathcal{A}$ for Π is *implementable* (read: "incentive-compatible") if there is a payment algorithm p that makes the following 3-step mechanism $M(\mathcal{A}, p)$ truthful: (1) request the agents' private information, receiving reports $\hat{t}_1, \ldots, \hat{t}_m$; (2) under the working hypothesis that $\hat{t}_i = t_i$ for every i, invoke $\mathcal{A}$ on the induced instance of Π and return the resulting outcome ω; (3) charge or distribute the payment $p_i(\omega, \hat{t}_1, \ldots, \hat{t}_m)$ to each agent i. Recall that such a mechanism is *truthful* if for every agent i and fixed reports by the other agents, reporting $\hat{t}_i = t_i$ is guaranteed to maximize i's utility $p_i(\omega, \hat{t}_1, \ldots, \hat{t}_m) - t_i(\omega)$ over all possible reports.[1] The key question above can be stated formally as: *can non-implementable polynomial-time algorithms for natural problems Π obtain better approximation ratios than implementable polynomial-time algorithms?*

This question remains wide open. The answer is known

[*]Department of Computer Science, Stanford University. Email: pdh@cs.stanford.edu. Supported in part by the ONR Young Investigator Award of the fourth author.

[†]School of Computer Science and Engineering, Hebrew University of Jerusalem. Email: shahard@cs.huji.ac.il. Supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities, and by a grant from the Israeli Academy of Sciences. This work was done while the author was visiting Stanford University.

[‡]Department of Computer Science, Stanford University. Email: shaddin@cs.stanford.edu. Supported in part by NSF grant CCF-0448664.

[§]Department of Computer Science, Stanford University. Email: tim@cs.stanford.edu. Supported in part by NSF CAREER Award CCF-0448664, an ONR Young Investigator Award, and an Alfred P. Sloan Fellowship.

[1]Other definitions of incentive-compatibility are possible, but this strong notion has been advocated widely for computational settings; see e.g. [20, Chapter 2] or [17] for detailed discussions.

to be "yes" for some natural problems, such as minimizing the sum of weighted completion times on parallel related machines, that cannot be approximated well even by implementable algorithms that are computationally unbounded [4]. Far more interesting are the problems that are optimally solvable via (computationally unbounded) implementable algorithms: these include $Q||C_{\max}$ [4], as well as arbitrarily general problems with a sum objective ($\sum_i t_i(\omega)$) [7, 10]. For NP-hard problems of this sort, *any separation between implementable and non-implementable polynomial-time algorithms must be conditional on $P \neq NP$*. Progress on this question in either direction is necessarily remarkable, both conceptually and technically: either incentive-compatibility imposes no additional difficulty for a massive class of important mechanism design problems, or else there is a non-trivial way of amplifying (conditional) complexity-theoretic approximation lower bounds using information-theoretic strategic requirements. The few results known along these lines are negative results for relatively complex "multi-parameter" problems [15, 19].

Single-Parameter Problems. This fundamental research issue is completely unresolved even in the important special case of *single-parameter agents*, for which a beautifully clean description of implementable algorithms was discovered by Archer and Tardos [4] (and earlier, in somewhat different contexts [16, 21]). Formally, a mechanism design problem is *single-parameter* if all outcomes are real m-vectors and agents' private cost functions have the form $t_i(\omega) = c_i\omega_i$ for a private real number c_i. $Q||C_{\max}$ can be phrased as a single-parameter problem, with ω denoting the work assigned to each machine and c_i the reciprocal of agent i's machine speed s_i. An algorithm for a single-parameter problem is *monotone* if increasing the value of a c_i (keeping other c_j's fixed) can only decrease the ith component of its solution. For $Q||C_{\max}$, monotonicity means that slowing down one machine can only decrease the work assigned to it by the algorithm. Archer and Tardos [4] proved that an algorithm for a single-parameter problem is implementable if and only if it is monotone. Conceptually, *polynomial-time single-parameter mechanism design is equivalent to polynomial-time monotone algorithm design*. Similarly, a randomized algorithm (for $Q||C_{\max}$, say) is implementable — i.e., via suitable payments, it can be extended to a mechanism in which truthful reporting always maximizes expected agent utility — if and only if the expected work assigned to a machine is nondecreasing in the machine speed (for fixed speeds of the other machines) [4].

The problem $Q||C_{\max}$ is the paradigmatic problem in single-parameter mechanism design (e.g. [14]), and was considered a realistic candidate problem for a conditional separation between implementable and non-implementable polynomial-time approximation algorithms. The problem admits an (exponential-time) implementable optimal algorithm, but all classical polynomial-time approximation algorithms for it, such as the polynomial-time approximation scheme (PTAS) designed by Hochbaum and Shmoys [11], are not monotone [4]. Archer and Tardos [4] devised a polynomial-time monotone randomized approximation algorithm that is 3-approximate with probability 1. Archer [3] later modified the algorithm and analysis to improve the performance guarantee to 2. While no superior guarantees have since been obtained for monotone algorithms, a sequence of papers have designed deterministic polynomial-time monotone algorithms with increasingly good approximation ratios; the current record, due to Kovács [13], is 2.8. (See also [1, 2, 5, 12].)

Results. Our main result is the first randomized monotone PTAS for the $Q||C_{\max}$ problem. The run-time bound and the approximation guarantee hold with probability 1; randomization is needed only for monotonicity. By applying the Archer-Tardos characterization and standard techniques to compute suitable payments in polynomial time, we obtain a polynomial-time, truthful (in expectation) mechanism whose approximation guarantee attains the best-possible one for polynomial-time algorithms (assuming $P \neq NP$).

The algorithmic techniques we develop for this result are flexible and easily yield additional new monotone algorithms for various single-parameter problems, such as: a deterministic quasipolynomial-time approximation scheme (QPTAS) for $Q||C_{\max}$ (improving over [2]); a randomized PTAS and deterministic QPTAS for minimizing the p-norm of loads on related machines; and a randomized PTAS for max-min scheduling on related machines (improving over [9]).

Techniques. We identify two key sources of non-monotonicity in classical approximation algorithms for $Q||C_{\max}$ and related problems, and develop a number of ideas to overcome them. Both the known PTASes for $Q||C_{\max}$ [8, 11] optimize over a compact but coarse representation of an allowable subset of schedules, represented as paths in a polynomial-size graph. This allowable subset fluctuates as a function of the machine speeds, so varying a machine speed causes unpredictable (and non-monotone) changes in algorithm behavior. Secondly, even when a machine speed perturbation leaves the allowable schedules invariant, attempting to optimize over their coarse representation inevitably yields only an approximate result. Approximation creates another opportunity for non-monotone behavior, with small perturbations in a machine speed potentially influencing the approximate solution chosen in an uncontrollable way. These difficulties are consistent with the empirical affinity between implementability and exact optimization over a fixed set that pervades the mechanism design literature.

Optimizing over a speed-dependent set of outcomes appears necessary to achieve a good approximation in polynomial time, and we begin by introducing a simple but powerful way to accomplish it safely: we first commit to a speed-independent set $\mathcal{X}$ of anonymous partitions of the jobs, then extend each partition to a concrete assignment of jobs to machines in a speed-dependent way, and conclude by exactly optimizing over this induced set of schedules. We prove that under a natural extension map for the second step, this three-step procedure is always monotone (for any set $\mathcal{X}$). This result is general and applies to many natural single-parameter problems.

The second and more technically challenging task is to identify a set $\mathcal{X}$ that is rich enough to contain near-optimal solutions for all possible machine speeds, yet structured enough to permit polynomial-time exact optimization. We use randomization twice to coax a job set into a form that allows a good compact representation. First, we artificially equalize the sizes of jobs that originally had similar sizes, randomly replacing them with the original job sizes at the end of the algorithm. Second, we allow fractional schedules, which we eventually convert to integral schedules via randomized rounding. Randomized rounding was also used by Archer and Tardos [3, 4], although our approximation target of $(1 + \epsilon)$ allows only the barest use of the technique: the jobs fractionally assigned to each machine must be dwarfed by those assigned fully. Nevertheless, we show that allowing even these highly restricted fractional job partitions permits an exact compact representation of a set that is guaranteed to include a near-optimal solution for every choice of machine speeds. Finally, our approach for optimizing over this set is inspired by the PTAS for $Q||C_{\max}$ of Epstein and Sgall [8], which in turn borrows some ideas from Hochbaum and Shmoys [11]; even here additional work is required, as our monotonicity constraint robs us of one of the degrees of freedom leveraged in [8], forcing us to enrich our representation.

2 Monotone Randomized PTAS for $Q||C_{\max}$

2.1 Monotone Algorithms via Smoothing, Rounding, and Exact Optimization

This section identifies a large class of monotone randomized algorithms for $Q||C_{\max}$, together with additional (strong) conditions that ensure an approximation ratio of $(1 + \epsilon)$. The next three sections design a polynomial-time algorithm that meets all of these requirements.

We leverage randomization in two distinct ways. First, for an arbitrary group S of k jobs, we define the *smoothed version* of S as a set of k jobs, each of size equal to the average size $(\sum_{j \in S} p_j)/k$ of a job of S. Given a schedule that includes these smoothed jobs, a *random shuffle* replaces each of them with a distinct job from S, with each such bijection equally likely. The smoothed size of a job is the same as its expected size following this random instantiation.

Second, we use the well-known technique of randomly rounding a fractional schedule. Precisely, a *fractional schedule* consists of a fractional assignment $\{y_{ij}\}_{i \in M}$ for each job j, where the y_{ij}'s are nonnegative and sum to 1 for each j. The *makespan* of a fractional schedule is defined as the value of the maximum (fractional) load $(\sum_j p_j y_{ij})/s_i$ of a machine i. By *randomly rounding* a fractional schedule, we mean that each job j is independently assigned to a machine, according to the probability distribution $\{y_{ij}\}_{i \in M}$. The expected work on a machine following randomized rounding equals the work assigned to it in the fractional schedule.

Finally, we require a technique to optimize over a speed-dependent set of allowable schedules without violating monotonicity. Every fractional schedule of n jobs to m machines induces an unordered *job partition* by ignoring the machine identities—a fractional partition of the n jobs into m classes, with each class corresponding to the job fractions assigned to a single machine. We sometimes call such a class S a *workload*, and use $|S|$ to denote the corresponding amount of work (sum of fractional job sizes). We use P_i to denote the class of a partition P with the ith-smallest amount of work (breaking ties arbitrarily). Given the speeds s of m machines, every job partition naturally defines $m!$ different fractional schedules, one for each bijection between workloads and machines. We single out the fractional schedule in which P_i is assigned to the ith slowest machine for each i, with ties between equal-speed machines broken in order of the machines' names, and call this the *schedule induced* by the given job partition and machine speeds. A trivial exchange argument shows that this is the "obvious" schedule to use given the workloads, in that it minimizes the makespan over the $m!$ possible schedules. Given m machine speeds, the *makespan* of a job partition is the makespan of the fractional schedule it induces. Our final technique for ensuring monotonicity is to optimize over some predetermined set $\mathcal{X}$ of (fractional) job partitions, evaluating each via the makespan of the induced schedule, breaking ties among optimal partitions according to a consistent ordering $\prec$.

We combine these three tools in the generic algorithm shown in Figure 1. The first step partitions the input jobs arbitrarily and applies job smoothing independently to each group. The smoothing step and the definitions of $\mathcal{X}$ and $\prec$ are required to be independent of s. The third step optimizes over the permissible partitions $\mathcal{X}$ with respect to s. The final two steps transform the induced fractional schedule of the smoothed jobs into an integral schedule of the original jobs via randomized rounding and random shuffling. A short

Input: n jobs with processing times $p_1, \ldots, p_n$ and m machines with speeds $s_1, \ldots, s_m$.

1. Group and smooth the jobs.
2. Define a set $\mathcal{X}$ of permissible fractional job partitions and a total ordering $\prec$ on $\mathcal{X}$.
3. Compute the partition in $\mathcal{X}$ with minimum makespan for s, breaking ties via $\prec$, and let σ_{frac} denote the induced schedule.
4. Transform σ_{frac} into an integral schedule σ_{smooth} of the smoothed jobs using randomized rounding.
5. Transform σ_{smooth} into an integral schedule of the original jobs using random shuffling.

Figure 1. A generic monotone algorithm. Only the third step is allowed to depend on s.

but slightly subtle proof shows that this algorithm is always monotone.

Lemma 2.1. *For every speed-independent job grouping and choice of $(\mathcal{X}, \prec)$, the randomized algorithm of Figure 1 is monotone.*

Proof. By the properties of randomized rounding and shuffling, the expected amount of work assigned to each machine equals the fractional work of the smoothed jobs assigned to it in the third step of the algorithm. Therefore, we only need to show that the fractional schedule of the smoothed jobs computed in the third step is monotone in the declared speeds.

Let $s = (s_i, s_{-i})$ and $\hat{s} = (\hat{s}_i, s_{-i})$ denote two speed vectors that differ only for machine i, with $s_i > \hat{s}_i$, and let $P, \hat{P} \in \mathcal{X}$ denote the corresponding optimal partitions. Let machine i be the kth slowest in s and the $\hat{k}$th slowest in $\hat{s}$, with $\hat{k} \leq k$. Monotonicity demands that $|\hat{P}_{\hat{k}}| \leq |P_k|$.

Let σ and $\hat{\sigma}$ denote the schedules induced by P for s and $\hat{s}$, respectively. If both schedules have the same makespan, then P is also a $\prec$-minimum optimal schedule for $\hat{s}$, so $\hat{P} = P$ and $|\hat{P}_{\hat{k}}| = |P_{\hat{k}}| \leq |P_k|$.

For the other case, call a machine *slow* if it is the ℓth slowest machine in $\hat{s}$ and is strictly slower than the ℓth slowest machine in s. The parameter ℓ lies in $\{\hat{k}, \ldots, k\}$ for each such machine. If the makespan of $\hat{\sigma}$ exceeds that of σ, then at least one slow machine, say the ℓth slowest in $\hat{s}$, determines the makespan in $\hat{\sigma}$. Since the schedule induced by $\hat{P}$ for $\hat{s}$ has makespan at most that of $\hat{\sigma}$ and $\ell \in \{\hat{k}, \ldots, k\}$, the proof is complete: $|\hat{P}_{\hat{k}}| \leq |\hat{P}_\ell| \leq |P_\ell| \leq |P_k|$. □

To control the approximation ratio of the generic algorithm in Figure 1, we impose three additional requirements — one for grouping, one for rounding, and one for the permissible job partitions. First, two jobs are *δ-comparable* if their processing times are within a $(1+\delta)$ factor of each other, and a *δ-grouping* of jobs is one in which only δ-comparable jobs are placed in a common group. Smoothing jobs via a δ-grouping can increase the makespan of an optimal schedule by at most a $(1+\delta)$ factor, and the worst-case damage caused by the random shuffling step is a $(1+\delta)$ increase in makespan. Second, a fractional job partition P is *δ-integral* if: (1) whenever a non-integral fraction of a job j belongs to some class P_i, $|P_i| \geq p_j/3\delta$; and (2) every class of P contains at most two fractional jobs. Randomly rounding a δ-integral schedule can only increase the makespan by a $(1+6\delta)$ factor. Finally, a set $\mathcal{X}$ of permissible job partitions is *δ-good* if, for every speed vector s, $\mathcal{X}$ contains a partition with makespan at most $(1+\delta)$ times that of an optimal integral schedule of the smoothed jobs. We have the following lemma.

Lemma 2.2. *Let δ be a sufficiently small positive constant. For every δ-grouping of jobs, every δ-good set $\mathcal{X}$ of δ-integral schedules, and every speed vector s, the schedule produced by the algorithm of Figure 1 has makespan $1 + O(\delta)$ times that of an optimal integral schedule, with probability 1.*

2.2 Permissible Partitions

This section identifies a δ-grouping and a δ-good set $\mathcal{X}$ of δ-integral job partitions for use in the generic algorithm (Figure 1). Consider n jobs, a parameter m, and a positive constant δ. We can assume that $1/\delta$ is a sufficiently large power of 2. We begin with our δ-grouping procedure, which leads to what we call *bucket smoothing*. Group together all jobs that share the same values of two parameters: the largest W that is a power of 2 with $p_j > \delta W$ (call it W^*); and the unique i such that the job size belongs to the *ith W^*-bucket*, defined as the interval $(\delta W^*(1 + (i-1)\delta), \delta W^*(1+i\delta)]$. Every such pair is δ-comparable, so this is a δ-grouping. By design, smoothing enforces the following property: whenever two jobs j, k lie in a common W-bucket, where W is some power of 2 with $W \geq p_j, p_k$ (possibly smaller than W^* above), the jobs have the same size and are thus interchangeable. (Other grouping schemes, such as buckets of the form $(\delta W^*(1+\delta)^{i-1}, \delta W^*(1+\delta)^i]$, work equally well.)

Bucket smoothing enables a succinct summary of the sizes of a set of jobs, described next. A *magnitude* is either 0 or a power of 2, and is used to determine the resolution at which we monitor job sizes. Let S denote a (possibly fractional) collection of smoothed jobs. If W is a magnitude at least the full size of each job of S, then the corresponding

W-configuration of S is a vector in which the first component (indexed by 0) denotes the total (fractional) work of the *W-small jobs* of S — those with full size at most δW — divided by δW; and each of the other $\approx \frac{1}{\delta^2}$ components i count the number of jobs of S whose full size lies in the ith W-bucket. Bucket-smoothing ensures that all (non-W-small) jobs in the same W-bucket have equal size.

We now build up the defining properties of the job partitions that we include in our set $\mathcal{X}$. Consider a fractional job partition P of the smoothed jobs, with P_i denoting the ith smallest class. Call an m-vector w of magnitudes *legal for P* if the following properties hold for each i:

(P1) w_i is a magnitude (0 or a power of 2) that is at least the full size of every job in P_i;

(P2) w_i is at least $1/\delta$ times the full size of every job that is fractionally assigned in P_i;

(P3) $|P_i| \in [\frac{1}{3}w_i, \frac{7}{6}w_i]$.

Property (P2) ensures that every non-w_i-small job of P_i is integral, and thus all components of the induced w_i-configuration except possibly the first are integral. Property (P3) ensures that there are no more than two legal values of w_i for a given P_i. Since the $|P_i|$'s are nondecreasing, legal w_i's must be *almost increasing* in the sense that $w_k \geq w_i/2$ whenever $k > i$.

If a partition P admits some vector w of legal magnitudes, and additionally each P_i contains at most two fractional jobs, then properties (P2) and (P3) together imply that P is δ-integral. The set of all such partitions is 0-good — for example, it includes all integral partitions — but appears far too rich to optimize over efficiently. This motivates our final properties, which impose just enough additional structure on the allowable job partitions to enable polynomial-time optimization without destroying δ-goodness.

Suppose w is legal for P. Place P_i in the *w_i-block* if $w_k \geq w_i$ for all $k > i$, and in the *$(w_i/2)$-block* if there is a $k > i$ with $w_k = w_i/2$. Since the w_i's are almost increasing, each block is a contiguous subset of the P_i's. The final class P_i of a W-block (necessarily with $w_i = W$) is the *block endpoint*. The largest class P_m is always a block endpoint.

A fractional job partition P is *permissible* if it is δ-integral and there are legal magnitudes w such that:

(P4) for every non-block endpoint P_i, the induced w_i-configuration of P_i is integral — i.e., the total (fractional) size of w_i-small jobs of P_i is a multiple of δw_i; and

(P5) for every block endpoint P_i other than P_m, the induced w_i-configuration of S_i is integral, where $S_i = \cup_{k\,:\,w_k \leq w_i} P_k$.

Thus most classes of a permissible partition have integral configurations, and the cumulative configurations at certain milestones (the block endpoints) are also integral. These properties are essential for the existence of a polynomial-size representation of permissible partitions.

We take $\mathcal{X}$ to be the set of permissible partitions, and order these partitions lexicographically by the work vector $(|P_1|, \ldots, |P_m|)$. (Ties between different partitions with the same work vector can be broken arbitrarily.) Neither the set $\mathcal{X}$ nor this ordering $\prec$ depend on the machine speeds. The next two sections give proofs of the following technical but important lemmas.

Lemma 2.3. *For every positive integer m, sufficiently small $\delta > 0$, and set of bucket-smoothed jobs, the corresponding set of permissible partitions is $O(\delta)$-good.*

Lemma 2.4. *For every constant $\delta > 0$, the problem of computing the permissible partition of a bucket-smoothed $Q||C_{\max}$ instance with minimum makespan, breaking ties via $\prec$, can be solved in polynomial time.*

Since permissible partitions are δ-integral, Lemmas 2.1–2.4 imply our main result.

Theorem 2.5. *There is a randomized monotone PTAS for $Q||C_{\max}$.*

After applying the Archer-Tardos characterization theorem [4] and standard techniques for efficiently computing suitable payments (see Section 2.5), Theorem 2.5 yields a polynomial-time, $(1+\epsilon)$-approximate, truthful in expectation mechanism for $Q||C_{\max}$.

2.3 Proof of Lemma 2.3

Fix $\delta > 0$, which we can assume is at most a sufficiently small constant, an instance of $Q||C_{\max}$ with bucket-smoothed jobs J and speed vector s, and an optimal schedule σ^*. Rename the machines so that $s_1 \leq s_2 \leq \cdots \leq s_m$. We extract from σ^* a permissible partition with makespan (with respect to s) at most $1 + O(\delta)$ times that of σ^*.

Let W_{max} denote the smallest power of 2 that upper bounds the work of every machine in σ^*. We first create a *reserve* $R \subseteq J$ for subsequent "rounding up" of fractional configurations. Assume without loss that the smallest job size is 1. For $W = 1, 2, 4, \ldots, W_{max}$ in turn, greedily add W-small jobs to R until the total size of R is at least $3\delta W$ (and at most $4\delta W$), or until there are no such jobs to add. We can assume that $|R| \geq 3\delta W_{max}$ at termination; otherwise one can check that re-assigning the jobs of R in σ^* to the machine with the most work yields a $(1+6\delta)$-approximate (integral) permissible partition. The proof plan is to begin with a set of legal weights, enforce properties (P4) and (P5), and finally restore δ-integrality;

each step preserves the properties already established while increasing the makespan by a $1 + O(\delta)$ factor.

Delete from σ^* all jobs of R and permute the workloads so that work is nondecreasing in machine speed. Let $S_1, \ldots, S_m$ denote the corresponding workloads, an ordered partition of $J \setminus R$ indexed by machine name. For each i, define w_i as the unique power of 2 with $w_i/2 < |S_i| \leq w_i$; these are legal for the job partition induced by the schedule. Also, $w_m = \max_i w_i$ since $|S_m| = \max_i |S_i|$, and $w_m \geq W_{max}/2$ since $|R| \leq 4\delta W_{max}$ (for δ sufficiently small). We repeatedly transform the schedule in what follows; by definition, the w_i's remain fixed at their initial values throughout the process.

Call machine i *non-integral* if $i \neq m$ and if the w_i-configuration of its current (possibly fractional) workload S_i is not integral — that is, the total (fractional) work created by the w_i-small jobs of S_i is not a multiple of δw_i. While there are two non-integral machines i and i', say with $w_i \leq w_{i'}$, we move w_i-small jobs from the former to the latter (where they are also $w_{i'}$-small), allowing fractional assignments, until one of the two machines becomes integral. This process terminates with at most one non-integral machine, say i. We conclude by moving w_i-small jobs from i to machine m — since $w_m = \max_i w_i$, they are also w_m-small — until the former becomes integral. This procedure terminates with a (fractional) schedule $(T_1, \ldots, T_m)$. Note that we *cannot* assume that $|T_1| \leq |T_2| \leq \cdots \leq |T_m|$. Nevertheless, this schedule induces a job partition meeting property (P4).[2] Since it alters the amount of work assigned to each machine i by less than δw_i and only reschedules small jobs, w remains legal for the job partition induced by the T_i's (for δ sufficiently small) and the makespan remains $(1 + O(\delta))$-approximate.

We dip into our reserve R to establish property (P5). We call a machine a *potential endpoint* if it is carrying a workload that would be a block endpoint in the job partition induced by the current schedule and w. Precisely, machine i is a potential endpoint of the current schedule $T_1, \ldots, T_m$ if $w_k > w_i$ for all machines k with $|T_k| > |T_i|$ or with $|T_k| = |T_i|$ and $i < k$. There is at most one potential endpoint per W-value. A potential endpoint i is *non-integral* if $w_i < w_m$ and the w_i-configuration of $\cup_{k : w_k \leq w_i} T_k$ is not integral. While there is a non-integral potential endpoint, we pick the one (i, say) with smallest W-value, and move w_i-small jobs from R to machine i, again permitting fractional assignments, until it becomes integral. Adding these jobs cannot create new potential endpoints and strictly decreases the number of non-integral potential endpoints. At termination, the job partition induced by the final schedule $(U_1, \ldots, U_m)$ and magnitudes w satisfies property (P5). Every machine to which we added jobs is a block endpoint of this partition, so the procedure does not violate (P4). Less than δw_i work is added to a non-integral potential endpoint i, so w remains legal and the makespan is increased by only a $1+O(\delta)$ factor. Also, R always contains enough jobs to implement each iteration: non-integrality of a potential endpoint i implies that not all w_i-small jobs are in R, so R began with at least $3\delta w_i$ units of w_i-small jobs; since W-values at least double each iteration, at most δw_i of these units were removed in prior iterations, leaving more than the requisite δw_i units available.

Once no non-integral potential endpoints remain, obtain the schedule $\hat{\sigma}$ by assigning all remaining jobs of R (of total size between $2\delta W_{max}$ and $4\delta W_{max}$) to machine m; this destroys neither the legality of w nor the $1 + O(\delta)$ approximation factor. Since $|S_m| = \max_i |S_i|$ and both job re-assignment procedures add at most δw_i work to i without removing jobs from m, $|U_m| > \max_i |U_i| - 2\delta W_{max}$. Thus m has the most work in $\hat{\sigma}$. The induced job partition, together with the legal magnitudes w, satisfies (P4) and (P5).

To restore δ-integrality, remove the w-small jobs $V \subseteq J$ from $\hat{\sigma}$, sort them in order of nondecreasing size, and reassign them to machines in order of nondecreasing w_i so that the work assigned to each machine is the same as in $\hat{\sigma}$ (using fractional assignments only when needed). Call this final schedule σ. The makespan is obviously unchanged. One easily checks that all jobs of V remain small for their assigned machine(s) in σ, so legality of w and properties (P4), (P5) are preserved. Finally, a job of V is fractionally assigned in σ only if it is the final small job re-assigned to one machine and the first to another. Since every machine has at most two (small) fractionally assigned jobs in σ, the schedule induces a permissible partition.

2.4 Proof of Lemma 2.4

This section shows that the problem of optimizing over permissible partitions can be formulated as a shortest-path computation in a layered network of polynomial size. We first describe the network, including motivation for its ingredients, and then sketch the precise correspondence between permissible partitions and certain paths in this network. Lemma 2.4 then follows easily.

Fix $\delta > 0$, m, and a set J of bucket-smoothed jobs. The graph G has $m + 2$ layers; the first (0) and last $(m + 1)$ contain only the source s and sink t, respectively. For $i \in \{1, 2, \ldots, m\}$, the ith layer will consist of a polynomial number of vertices, each endowed with six labels. An arc from layer i to $i + 1$ is meant to dictate the ith-smallest workload of a permissible partition and the corresponding magnitude w_i, as well as the value W of the W-block to which the $(i + 1)$th workload belongs (for $i + 1 \leq m$). Our set of vertex labels will be rich enough so that the intentions

[2] Strictly speaking, this holds provided $|T_m| > \max_{i<m} |T_i|$; as we'll see, the mth workload will satisfy this property by the end of the proof.

of an arc can be inferred from the labels of its endpoints.

Precisely, every vertex in a layer $i \in \{1, 2, \ldots, m\}$ is labeled with two magnitudes W_1 and W_2. Each is required to be either 0 or in a polynomial-size set of powers of 2, ranging from the smallest power of two that upper bounds $p_{\min}$ to the smallest one that upper bounds $np_{\max}$, where $p_{\min}$ and $p_{\max}$ denote the smallest and largest job sizes. We also insist that $W_2 \geq 2W_1$. Choices of W_1, W_2 that meet these constraints are called *valid*. These labels are meant to indicate that the ith workload belongs to the W_2-block — and thus its magnitude will be either W_2 or $2W_2$ — while the previous distinct block is the W_1-block.

The other four vertex labels A_1, B_1, A_2, B_2 summarize the sizes of the jobs assigned to the first $i - 1$ workloads. Each is constrained to be an integral W-configuration for some magnitude W; there are only polynomially many ($n^{O(1/\delta^2)}$) such configurations. Configuration A_1 is meant to be the W_1-configuration of the set of jobs assigned to previous workloads $k < i$ with $w_k \leq W_1$; B_1 the $2W_1$-configuration of jobs in previous workloads $k < i$ in the W_1-block with $w_k = 2W_1$; A_2 and B_2 the W_2- and $2W_2$-configurations of jobs in previous workloads $k < i$ in the (current) W_2-block with $w_k = W_2$ and $w_k = 2W_2$, respectively. Four distinct labels are required to faithfully capture properties (P4) and (P5) of permissible partitions as integrality constraints on configurations.

Our intents for the labels A_1, B_1, A_2, B_2 suggest the following additional constraints. To explain them, recall that a W-configuration has a component (indexed by 0) indicating the total (possibly fractional) size of W-small jobs, divided by δW; and $\approx 1/\delta^2$ components that count the number of jobs in each W-bucket. We call a W-configuration C *realizable* if the total size of the W-small jobs of J is at least $C_0 \cdot \delta W$, and for each $i > 0$, at least C_i jobs of J belong to the ith W-bucket. Next, note that a W-configuration can be uniquely rewritten as a W'-configuration at a coarser resolution $W' \geq W$, with jobs moving to lower-indexed buckets, and some non-W-small jobs becoming W'-small. Thus two configurations with different magnitudes can be sensibly added to produce one at the larger magnitude (though the sum of two integral configurations can have a fractional first component). Finally, we call the parameters $W_1, W_2, A_1, B_1, A_2, B_2$ *valid* if W_1, W_2 are valid, and every subset of A_1, B_1, A_2, B_2 sums to a realizable configuration (at the appropriate magnitude W_1, $2W_1$, W_2, or $2W_2$). Every layer $i \in \{1, 2, \ldots, m\}$ of G has one vertex for every possible set of valid parameters.

Next we describe the edge set of G, beginning with the edges from layer i to $i + 1$ for $i \in \{1, 2, \ldots, m - 1\}$. Let $(W_1, W_2, A_1, B_1, A_2, B_2)$ be the (valid) parameters of a vertex u in layer i. Let N_u denote the vertices v of layer $i+1$ that meet one of the following three conditions:

(A) all of v's parameters match those of u except for its fifth parameter, which is some configuration that is (componentwise) at least A_2;

(B) all of v's parameters match those of u except for its sixth parameter, which is some configuration that is (componentwise) at least B_2;

(C) v's parameters are $(W_2, W_3, D, B_2, 0, 0)$, where D is some integral W_2-configuration that is (componentwise) at least the (possibly fractional) W_2-configuration $A_1 + B_1 + A_2$.

These three cases are meant to correspond to the following scenarios: (A) workload $i + 1$ also belongs to the (current) W_2-block and $w_i = W_2$; (B) workload $i + 1$ also belongs to the (current) W_2-block but $w_i = 2W_2$; and (C) workload $i + 1$ belongs to the W_3-block for some $W_3 > W_2$. In all three cases, we can extract from the labels of u, v a proposed magnitude w_{uv} for the ith workload — W_2 in (A) and (C), $2W_2$ in (B) — and a corresponding w_{uv}-configuration, which we can interpret as a proposed ith workload: in (A), the increase in the fifth parameter; in (B), the increase in the sixth parameter; and $D - A_1 - B_1 - A_2$ in (C). For $v \in N_u$, let x_{uv} denote the amount of work represented by the corresponding w_{uv}-configuration C. Because the jobs J are bucket-smoothed, x_{uv} is uniquely defined as $C_0 \cdot \delta w_{uv}$ plus $\sum_{h>0} C_h \cdot z_h$, where z_h denotes the common size of every job that lies in the hth w_{uv}-bucket. Eying constraint (P3), we connect vertex u to every $v \in N_u$ for which $x_{uv} \in [\frac{1}{3}w_{uv}, \frac{7}{6}w_{uv}]$. We assign each such edge (u, v) a length of x_{uv}. We classify such edges as *type A*, *type B*, or *type C* according to the condition met by its endpoints' labels.

The edges incident to s and t are defined similarly. The source is connected to all vertices v of layer 1 that possess a label in which all parameters but the second are zero; such an edge effectively determines the value of W for the first W-block, but does not determine any workloads. These edges are all assigned a length of zero and have no type. Finally, consider a node v of layer m with valid parameters $(W_1, W_2, A_1, B_1, A_2, B_2)$. We adopt W_2 as the proposed magnitude for the mth workload. We connect v to t in G if and only if there is a realizable W_2-configuration C such that $A_1 + B_1 + A_2 + B_2 + C$ is the $2W_2$-configuration of the full set J of jobs, and the corresponding amount of work x_{vt} of C lies in $[\frac{1}{3}W_2, \frac{7}{6}W_2]$. (There can be more than one such configuration C, but all solutions represent the same amount of work.) Each such edge (v, t) is assigned a length of x_{vt} and is classified as a type C edge. This construction of the network G can be performed in polynomial time.

We now verify that our construction represents permissible partitions.

Lemma 2.6. *Let G denote the network corresponding to a bucket-smoothed instance of $Q||C_{\max}$ and a constant $\delta > 0$.*

(a) For every permissible partition P, there is an s-t path of G whose sequence of edge lengths is $0, |P_1|, |P_2|, \ldots, |P_m|$.

(b) Given an s-t path of G whose sequence of edge lengths is $0 \leq x_1 \leq x_2 \leq \cdots \leq x_m$, a permissible partition P with $|P_i| = x_i$ for every i can be constructed in polynomial time.

Proof. Consider a permissible partition P and corresponding legal weights w. For each $i \in \{1, \ldots, m\}$, P and w naturally induce a vertex $v_i = (W_1, W_2, A_1, B_1, A_2, B_2)$ of layer i of G: W_1, W_2 are defined so that P_i belongs to the W_2-block of P and the previous distinct block is the W_1-block (or $W_1 = 0$ if no such block exists); and A_1, B_1, A_2, B_2 are derived from P according to their intended meanings, discussed above. Since P satisfies properties (P4) and (P5), all four configurations are integral. By definition and properties (P1)–(P3), $W_1, W_2, A_1, B_1, A_2, B_2$ are valid parameters, corresponding to some vertex v_i of layer i of G. The edge (s, v_1) is clearly present in G. Our definition of edge lengths in G ensures that $x_{v_i,v_{i+1}}$ equals the work $|P_i|$, so property (P3) implies that the edges $(v_1, v_2), (v_2, v_3), \ldots, (v_m, t)$ are present in G. The sequence of edge lengths along this path is precisely $0, |P_1|, |P_2|, \ldots, |P_m|$.

Conversely, consider an s-t path of G with intermediate vertices $v_1, \ldots, v_m$ and a nondecreasing sequence of edge lengths. As outlined above, the edges of this path suggest magnitudes w and, for each i, a corresponding w_i-configuration C^i. (Recall that C^m can be inferred from $C^1, \ldots, C^{m-1}$ and the set J of all jobs.) For example, if the label of v_i is $(W_1, W_2, A_1, B_1, A_2, B_2)$, (v_i, v_{i+1}) is a type-A edge, and the fifth parameter of v_{i+1}'s label is A'_2, then we define $w_i = W_2$ and $C^i = A'_2 - A_2$. The components of C^i other than the first indicate how many jobs from the different w_i-buckets should be (integrally) assigned to the ith workload, while the first component of C^i describes the total fractional size of w_i-small jobs that should be assigned to this workload. Our realizability constraints ensure that these configurations can be translated into a job partition $P_1, \ldots, P_m$ in the obvious way, with the final small job assignments performed as in the last step of Section 2.3 — in nondecreasing order of workload magnitude and of job size, resorting to fractional assignments only when needed. This translation can be performed in polynomial time, ensures that $|P_i| = x_{v_i v_{i+1}}$ for every i, and enforces δ-integrality. The produced partition P clearly satisfies properties (P1) and (P2) with respect to magnitudes w. The definition of the edge set of G ensures that P and w satisfy (P3). For property (P4), observe that every fractional configuration C^i results from a type C edge (v_i, v_{i+1}), and the corresponding workload P_i is necessarily a block endpoint of P with respect to w. To complete the proof, note that every block endpoint P_i arises from some type C edge (v_i, v_{i+1}), and property (P5) then follows immediately from the integrality of the third parameter of v_{i+1} (representing the jobs assigned to workloads $k \leq i$ with $w_k \leq w_i$). $\square$

To prove Lemma 2.4, consider a bucket-smoothed $Q||C_{\max}$ instance and a constant $\delta > 0$. Rename machines so that $s_1 \leq s_2 \leq \cdots \leq s_m$. Form the (speed-independent) network representation G of permissible partitions, and assign a cost of x_{uv}/s_i to every edge (u, v) traveling from layer i to layer $i+1$. By Lemma 2.6, computing the permissible partition with minimum makespan for s polynomial-time reduces to computing the s-t path of G that has a nondecreasing sequence of edge lengths and minimizes the bottleneck edge cost (breaking ties among optimal solutions lexicographically according to the vector of lengths x). This problem can be solved in polynomial time — either directly using dynamic programming, or via Dijkstra's algorithm following a simple graph transformation that eliminates s-t paths that are not nondecreasing.

2.5 Computing Payments

To extend our randomized monotone PTAS for $Q||C_{\max}$ to a truthful (in expectation) mechanism, we compute suitable payments by integrating the "work curve" of each machine as described in [4]. Accomplishing this in polynomial time requires two observations (see also [3, §2.6]). First, we can pre-round each machine speed down to the nearest power of $1+\delta$ before running our algorithm without affecting its monotonicity or PTAS guarantee (where δ is a suitably small constant). Second, let $p_{\min}$ and $p_{\max}$ denote the smallest and largest job sizes, respectively. The minimum non-zero expected load assigned to a machine by our algorithm is $p_{\min}$; this follows from properties (P2) and (P3) of permissible partitions, assuming δ is sufficiently small. The approximation guarantee of $(1+\epsilon)$ therefore ensures that no machine more than a $(1+\epsilon)np_{\max}/p_{\min}$ factor slower than the fastest receives non-zero work. This in turn implies that, with the pre-rounded speeds, there are only polynomially many potential breakpoints in the work curve of a machine i (given the fixed speeds of the other machines). Appropriate payments are then easy to compute in polynomial time.

3 Deterministic Algorithms and Additional Single-Parameter Problems

3.1 Monotone Deterministic QPTAS for $Q||C_{\max}$

The importance of randomization in our monotone PTAS for $Q||C_{\max}$ (Theorem 2.5) is evident and we leave open the question of whether or not a deterministic monotone PTAS exists. Nevertheless, we can apply our generic algorithm (Figure 1) to obtain easily a monotone deterministic quasi-polynomial-time approximation scheme (QPTAS) for the problem.

Theorem 3.1. *There is a deterministic monotone QPTAS for* $Q||C_{\max}$.

Proof. Fix a set of n jobs and parameters m, δ, and set $l = \lceil \log_{(1+\delta)}(m/\delta) \rceil$. Let $\mathcal{S}$ denote the nondecreasing speed vectors s with $s_m = 1$ and with each s_i either 0 or of the form $(1+\delta)^{-k_i}$ for an integer k_i between 0 and l. There is a quasi-polynomial number $m^{O(l)}$ of such speed vectors. Compute a $(1+\delta)$-approximate schedule for each using a (non-monotone) PTAS for $Q||C_{\max}$ such as [11] or [8], and let $\mathcal{X}$ denote the induced set of (integral) job partitions. We can explicitly construct and optimize over $\mathcal{X}$ in quasi-polynomial time. Order $\mathcal{X}$ lexicographically by sorted work vectors, as in Theorem 2.5.

By Lemmas 2.1 and 2.2, we can complete the proof by showing that $\mathcal{X}$ is $O(\delta)$-good. Consider an arbitrary speed vector s; renaming and scaling, we can assume that s is nondecreasing with $s_m = 1$. Call machine i *slow* if $s_i < \delta/m$. Obtain the speed vector $\hat{s}$ by zeroing out the speeds of slow machines and rounding all other speeds down to the nearest integer power of $(1+\delta)^{-1}$. The optimal makespan for machines speeds $\hat{s}$ is at most $1 + O(\delta)$ times that for s (take an optimal schedule for s and reassign jobs on slow machines to machine m). By construction, $\mathcal{X}$ contains a partition inducing a $(1+\delta)$-approximate schedule for $\hat{s}$; this schedule is $(1+O(\delta))$-approximate for s. $\square$

The exponential dependence on $\log^2 m$ in Theorem 3.1 improves upon the exponential dependence on m in the deterministic monotone algorithm in [2].

3.2 Further Results

We can also extend our results to other parallel related machine scheduling problems. We mention two: minimizing the p-norm of the machine loads (for $p \in [1, \infty]$) and maximizing the minimum machine load. Both problems admit a non-monotone PTAS [6, 8]. The obvious modification of our generic algorithm (Figure 1), in which we replace the makespan objective in the third step by the appropriate objective, remains monotone for these two problems. The proof of Lemma 2.1 extends easily to max-min scheduling; for minimizing the p-norm of the machine loads, some modifications are required.

Proof. (of Lemma 2.1, adapted to minimizing the p-norm.) Let $s = (s_i, s_{-i})$ and $\hat{s} = (\hat{s}_i, s_{-i})$ denote two speed vectors that differ only for machine i, with $s_i > \hat{s}_i$, and let $P, \hat{P} \in \mathcal{X}$ denote the corresponding optimal partitions. Let machine i be the kth slowest in s and the $\hat{k}$th slowest in $\hat{s}$, with $\hat{k} \le k$. Assume for contradiction that $|P_k| < |\hat{P}_{\hat{k}}|$, so $|P_\ell| \le |P_k| < |\hat{P}_{\hat{k}}| \le |\hat{P}_\ell|$ for each $\ell \in \{\hat{k}, \ldots, k\}$.

Let $s(\ell), \hat{s}(\ell)$ denote the speeds of the ℓth slowest machines in s and $\hat{s}$, respectively. Switching from speeds s to $\hat{s}$ increases the pth power of the p-norm of the schedule induced by P by $\sum_{\ell=\hat{k}}^{k} |P_\ell|^p(\hat{s}(\ell)^{-p} - s(\ell)^{-p})$ and that of the schedule induced by $\hat{P}$ by $\sum_{\ell=\hat{k}}^{k} |\hat{P}_\ell|^p(\hat{s}(\ell)^{-p} - s(\ell)^{-p})$. Thus the p-norm of the latter schedule increases at least as much as the former, contradicting the assumption that $\hat{P}$ is the $\prec$-minimum optimal schedule for $\hat{s}$. $\square$

For minimizing the p-norm of the machine loads, Theorem 3.1 now carries over without change. To extend Theorem 2.5 to this problem, we define permissible partitions as before. Lemma 2.3 remains valid because its proof extracts a permissible partition from an arbitrary integral schedule while increasing the work assigned to each machine, and hence the p-norm, by a $1 + O(\delta)$ factor. Lemma 2.4 requires only trivial modifications and truthful payments can be computed as in Section 2.5.

Extending our randomized monotone PTAS to max-min scheduling requires much more work. The difficulty is that removing jobs from machines, as in the reservation procedure in the proof of Lemma 2.3, can destroy near-optimality. To circumvent this problem, we relax our definition of permissible partitions. First, we insist only on 2δ-integrality rather than δ-integrality. For magnitudes w to be legal for a partition P, we only require the final magnitude w_m to be at least $1/2\delta$ times the full size of every job fractionally assigned in P_m (cf., property (P2)). Finally, we replace property (P5) by:

(P5') for every block endpoint P_i other than P_m, either the induced w_i-configuration of S_i is integral, or else S_i includes all w_i-small jobs, where $S_i = \cup_{k\,:\,w_k \le w_i} P_k$.

We next modify the proof of Lemma 2.3 to show that this relaxed set of permissible partitions is $O(\delta)$-good for the max-min objective.

Proof. (of Lemma 2.3, adapted to max-min scheduling.) Fix a job set, machine speeds s, and an optimal schedule σ^*. Let $(S_1, \ldots, S_m)$ denote the corresponding workloads. We can assume that $s_1 \le \cdots \le s_m$ and $|S_1| \le \cdots \le |S_m|$. We extract from σ^* a permissible partition with minimum load (with respect to s) at least $1 - O(\delta)$ times that of σ^*.

For each i, define w_i as the unique power of 2 with $w_i/2 < |S_i| \le w_i$; these are legal for the job partition induced by the schedule. We begin by iterating through the magnitudes W occurring in w in increasing order. We fractionally re-assign W-small jobs from machines with magnitude larger than W to those with magnitude equal to W, until the total fractional amount of W-small jobs assigned to the latter machines is either a multiple of δW or is all W-small jobs. This procedure enforces a strengthened form of property (P5'), and it removes at most δw_i work from each machine i (at most δW in each iteration with $W < w_i$).

To establish (P4), we again iterate through the magnitudes W of w, in arbitrary order. As in the proof of Lemma 2.3, we can re-assign W-small-jobs between machines with magnitude W until only one such machine remains with a non-integral W-configuration. Re-assigning again if needed, we can assume that this machine is the most heavily loaded one with magnitude W (and thus will be a block endpoint provided it winds up in the W-block). These re-assignments do not affect any previously established properties. The job partition induced by the resulting schedule satisfies property (P4), except for machines i that belong to the $(w_i/2)$-block of the partition and are the most heavily loaded machine with magnitude w_i. For each such machine i, we re-assign the minimal (fractional) amount of w_i-small jobs to the next block endpoint with magnitude exceeding w_i. This is always possible unless $w_i = \max_k w_k$; in this case, we move the same amount to the most heavily loaded machine k; our relaxed version of property (P2) allows this, even in the event that w_k is only $w_i/2$. No machine i loses more than δw_i work in this second round of re-assignments.

Finally, we restore 2δ-integrality of the job partition by re-assigning small jobs as at the end of Section 2.3. $\square$

Modifying the representation and algorithm in Section 2.4 to accommodate this wider set of permissible partitions is relatively straightforward. Parameters $W_1, W_2, A_1, B_1, A_2, B_2$ are now *valid* if each of B_1, A_2, B_2 is integral, A_1 is either integral or represents a superset of all W_1-small jobs, and all subsets of $\{A_1, A_2, B_1, B_2\}$ sum to realizable configurations. Crucially, there are still only polynomially many valid sets of parameters. As in Section 2.4, given machine speeds, the optimal permissible partition can be found by an s-t path computation.

Theorem 3.2. *There is a randomized monotone PTAS for maximizing the minimum load on related machines.*

This improves over a recent result of Epstein and van Stee [9], who obtain a (deterministic) monotone PTAS for a constant number of machines. Truthful payments can be computed as in Section 2.5. Unlike all other problems studied in this paper, however, the form of the max-min objective (where a finite approximation algorithm must assign non-zero work to every machine) implies that such payments cannot be chosen to satisfy individual rationality (see [4]).

References

[1] P. Ambrosio and V. Auletta. Deterministic monotone algorithms for scheduling on related machines. In *WAOA '04*, pages 267–280.

[2] N. Andelman, Y. Azar, and M. Sorani. Truthful approximation mechanisms for scheduling selfish related machines. *Theory Comput. Syst.*, 40(4):423–436, 2007.

[3] A. Archer. *Mechanisms for Discrete Optimization with Rational Agents*. PhD thesis, Cornell University, 2004.

[4] A. Archer and É. Tardos. Truthful mechanisms for one-parameter agents. In *FOCS '01*, pages 482–491.

[5] V. Auletta, R. D. Prisco, P. Penna, and G. Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In *STACS '04*, pages 608–619.

[6] Y. Azar and L. Epstein. Approximation schemes for covering and scheduling on related machines. In *APPROX '98*, pages 39–47.

[7] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

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

[9] L. Epstein and R. van Stee. Maximizing the minimum load for selfish agents. In *LATIN '08*, pages 264–275.

[10] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

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

[12] A. Kovács. Fast monotone 3-approximation algorithm for scheduling related machines. In *ESA '05*, pages 616–627.

[13] A. Kovács. Tighter approximation bounds for LPT scheduling in two special cases. In *CIAC '06*, pages 187–198.

[14] R. Lavi. Computationally efficient approximation mechanisms. In N. Nisan, T. Roughgarden, É. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 12, pages 301–329. Cambridge University Press, 2007.

[15] R. Lavi, A. Mu'alem, and N. Nisan. Towards a characterization of truthful combinatorial auctions. In *FOCS '03*, pages 574–583.

[16] R. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6:58–73, 1981.

[17] N. Nisan. Introduction to mechanism design (for computer scientists). In N. Nisan, T. Roughgarden, É. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 9, pages 209–241. Cambridge University Press, 2007.

[18] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1/2):166–196, 2001.

[19] C. H. Papadimitriou, M. Schapira, and Y. Singer. On the hardness of being truthful. In *FOCS '08*.

[20] D. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, 2001.

[21] J. Riley and W. Samuelson. Optimal auctions. *American Economic Review*, 71:381–92, 1981.