The Power of Verification for One-Parameter Agents ⋆

Vincenzo Auletta, Roberto De Prisco,
Paolo Penna *, and Giuseppe Persiano

Dipartimento di Informatica ed Applicazioni
"R.M. Capocelli"
Università di Salerno
via ***S. Allende*** *2*
I-84081 Baronissi (SA)
Italy

Abstract

We initiate the study of mechanisms with *verification* for *one-parameter* agents. We give an algorithmic *characterization* of such mechanisms and show that they are *provably better* than mechanisms without verification, i.e., those previously considered in the literature. These results are obtained for a number of optimization problems motivated by the Internet and recently studied in the algorithmic mechanism design literature. The characterization can be regarded as an alternative approach to existing techniques to design *truthful* mechanisms. The construction of such mechanisms reduces to the construction of an algorithm satisfying certain "monotonicity" conditions which, for the case of verification, are much less stringent. In other words, verification makes the construction easier and the algorithm more efficient (both computationally and in terms of approximability).

Key words: Algorithmic game theory, mechanism design, theory of algorithms

⋆ Research funded by the European Union through IST FET Integrated Project AEOLUS (IST-015964).

* Corresponding author.
Email addresses: auletta@dia.unisa.it (Vincenzo Auletta), robdep@dia.unisa.it (Roberto De Prisco), penna@dia.unisa.it (Paolo Penna), giuper@dia.unisa.it (Giuseppe Persiano).

1 Introduction

The emergence of the Internet as the platform for distributed computing has posed interesting questions on how to design efficient solutions which account for the lack of a "central authority" [5,9,10]. This aspect is certainly a key ingredient for the success of the Internet and, probably, of any "popular" system that one can envision (peer-to-peer systems are a notable example of this type of anarchic systems). In their seminal works, Koutsoupias and Papadimitriou [5] and Nisan and Ronen [9] suggest a *game-theoretic* approach in which the various "components" of the system are modeled as *selfish agents*: each agent performs a "strategy" which results in the highest *utility* for herself. For instance, each agent may control a link of a communication network and each link has a cost for transmitting (i.e., for using it). This cost depends on the *type* of the link and a protocol that wishes to establish a minimum-cost path between two nodes would have to ask the agents for the type of the corresponding link [9,2]. An agent may thus have a higher utility by misreporting her type (e.g., untruthfully report that her link is of a "very slow" type in order to induce the protocol to forward packets through "somebody else's" link or to receive a higher payment in reward). Nisan and Ronen [9] propose a *mechanism design* approach that combines an underlying algorithm (e.g., a shortest-path algorithm) with a suitable payment function (e.g., how much we pay an agent for using her link). The payment rule is used to ensure that no agent will find convenient to misreport her type. One of the strongest and most studied solution concepts in game theory is that of *truthful mechanism*: every agent can maximize her own utility simply by reporting truthfully her type, no matter which strategy the other agents follow. In other words, truth-telling is a *dominant strategy* for all agents. The main interest in truthful mechanisms stems from the need of "protocols" which do not assume an "altruistic" behavior of the involved parties [9]. In an ideal scenario, the involved parties will never "cheat" and thus some underlying algorithm will be able to compute a "globally" optimal solution. In practice the algorithm runs on inputs provided by the selfish agents. This motivates the need for truthful mechanisms to make sure that the algorithm gets the "true" input. Unfortunately, not every algorithm can be turned into a truthful mechanism. Indeed, a central question is (algorithmic) game theory is:

Which algorithms can be turned into a truthful mechanism?

This question can be rephrased in terms of combinatorial optimization problems and algorithms that solve them optimally or approximately [9,2]:

Which problems admit optimal truthful mechanisms?

What is the best approximation ratio of a truthful mechanism?

These questions have been raised first by Nisan and Ronen [9] who considered the "selfish version" of the classical problem of scheduling unrelated machines: each machine belongs to a selfish agent who reports the type of the machine, that is, how long it will take to execute different jobs on that machine. For this problem, they showed that no truthful mechanism can achieve an approximation factor better than 2, in spite of the fact that $(1+\varepsilon)$-approximations can be achieved by polynomial-time algorithms for every $\varepsilon > 0$. Moreover, existing mechanism design techniques guarantee only n-approximations, with n being the number of machines. Nisan and Ronen [9] then introduce a new paradigm called *mechanisms with verification*: the idea is that the mechanism can observe the jobs completion times and gain some "partial" information on the type of the machines. These additional information allows for more powerful mechanisms, since they can guarantee *exact* solutions for this scheduling problem. So, a natural question that arises is whether verification helps for other problems involving selfish agents:

What is the power of verification?

This work addresses precisely this question. In particular, we apply the basic idea of Nisan and Ronen [9] to an important and well-studied class of problems with *one-parameter* agents [8,2]. Our results relate to all questions above since we provide a *characterization* of truthful mechanisms with verification for very general settings. From this result we obtain truthful mechanisms with verification for several optimizations problems considered in the literature. The approximation ratio of these mechanisms outperforms the best known mechanisms which do not use verification. Moreover, for several of these problems, the approximation ratio achievable with verification is provably better than the approximation ratio that any mechanism "without verification" can achieve. Finally, our characterization explains in a simple and natural way *why* mechanisms with verification are more powerful for one-parameter agents. Indeed, truthful mechanisms (without verification) are characterized by so called *monotone* algorithms [8,2]. Mechanisms with verification relax this condition and allows to use what we call in this paper *roughly monotone* algorithms. For several problems, roughly monotone algorithms can achieve optimal solutions, while the best monotone algorithm must have a worse approximation factor.

Our interest in one-parameter agents is twofold. First, since these problems have been well understood for the case without verification, the study of mechanisms with verification in this setting gives us precise indications about the "power of verification". (Namely, that we need only a weaker condition on the algorithm, thus making the construction of a mechanism simpler and more efficient for certain problems.) Second, there are several applications which can be easily modelled in terms of one-parameter agents and naturally fit in the model of mechanisms with verification. These include, grid computing applications in which jobs have to be executed on machines whose "speed" is

only known to the owners [2] (a simpler version of the scheduling problem in [9] motivating mechanisms with verification), BGP routing when Autonomous Systems (AS) have subjective "per-packet" costs [7] (the routing policy determines the amount of traffic that transits through each AS and packets latency – i.e., the cost for the AS – can be observed).

1.1 Mechanisms with verification and one-parameter agents

Nisan and Ronen [9] introduced mechanisms with verification in the following scenario. We have a number of jobs and a number of machines. Each machine i is of some *type* t_i meaning that the execution of a job j on a machine i requires time t_i^j. Each machine corresponds to a *selfish agent* and the type of machine i is only known to agent i. The key idea of mechanisms with verification has been suggested by Nisan and Ronen [9] for this problem: machine i cannot release job j before t_i^j time steps. Therefore, if agent i reports a type b_i^j for job j and a job j is assigned to machine i, the mechanism is able to detect that b_i^j is *not* the true type i if $b_i^j < t_i^j$. If that is the case, and only in this case, the mechanism can penalize this agent. We apply this simple idea to the case of *one-parameter* agents studied by Archer and Tardos [2]. Here, each feasible solution x assigns some amount of work $w(x, i)$ to agent i who, given her type t_i, must spend $w(x, i) \cdot t_i$ time steps for completing the given work. Agent i can strategically misreport her type to some other value b_i and release the results of its computations after its completion time (or later). So, agent i is not able to complete her work consistently with her reported type if and only if $w(x, i) \cdot b_i < w(x, i) \cdot t_i$. In this case, we say that agent i is *caught lying* and the mechanism can penalize her by denying the corresponding payment. A *mechanism with verification* provides a payment to agent i if and only if agent i is *not caught lying*. That is, if and only if $b_i \cdot w(x, i) \geq t_i \cdot w(x, i)$, where x is the solution chosen by the algorithm given that agent i reports a type b_i and her (true) type is t_i. In contrast, mechanisms without verification *always* provide the payment to agent i. A natural problem involving one-parameter agents is the *scheduling on related machines* [2]: jobs have weights and machines have speeds, and an assignment determines the amount of work allocated to each machine. In Section 2 we provide a formal setting for mechanisms with verification in the case of one-parameter agents.

1.2 Overview of the results

One-parameter agents provide a very natural setting for representing subjective preferences when agents receive different amounts of the same "good" [8]: each agent valuates a single unit some amount t_i and her total valuation is

this value times the total amount that she gets. In the scenario proposed by Archer and Tardos [2], goods are actually units of work and the type of each agent is the amount of time it costs for such an agent to complete one unit of work. Both of these works show that truthful mechanisms without verification are characterized by *monotone* algorithms [8,2]: intuitively speaking, no agent receives more work if she increases the reported cost per unit, i.e., reporting b_i will cause this agent to get at most the same amount of work as if reporting any $b'_i > b_i$.

We first give a *characterization* of truthful mechanisms with verification for one-parameter agents in the case of finite domains (i.e., there is a finite set of values that an agent can report). In particular, we show that there is a truthful mechanism using some algorithm if and only if the latter is *roughly monotone*: if an agent receives no work when reporting b_i, then it also receives no work when reporting any $b'_i > b_i$. We stress that the "if part" of our characterization is constructive and that the hypothesis of finite domains can be easily removed for many "natural" problems. We indeed provide sufficient conditions for obtaining *computationally efficient* truthful mechanisms with verification. This result yields a new mechanism design technique which essentially turns polynomial-time roughly monotone algorithms into polynomial-time truthful mechanism with verification having "roughly" the same approximation guarantee. So, the construction of the entire mechanism reduces to the construction of a roughly monotone algorithm for the problem at hand. This is an improvement over existing approaches [8,2] since, as mentioned above, the rough monotonicity condition is less stringent than the monotonicity one. We indeed apply this new approach to a number of problems previously considered in the literature and obtain improvements over existing mechanisms.

Our first application in one of the most studied and intriguing problems in algorithmic mechanism design: scheduling jobs on *selfish machines* in order to minimize the *makespan* [9,2,1,6]. We give the first truthful *polynomial-time* $(1+\varepsilon)$–approximation mechanism for *any number* of related machines having their execution times bounded by some constant, for every $\varepsilon > 0$. Notice that the mechanism for unrelated machines in [9] also assumes bounded execution times, but works only for a finite number of machines. Similarly, the mechanism by Andelman *et al.* [1], which does not use verification, also requires a constant number of machines. For arbitrary number of machines, the best approximation factor achieved by a polynomial-time truthful mechanism is 3 which is achieved by Kovács [6], while an exact *not* polynomial-time truthful mechanism is given in [2]. Neither mechanism uses verification.

We then show that mechanisms with verification are *provably better* than mechanisms without verification for several variants of the above problem considered in the literature. Our upper bounds "break" the inapproximability results for mechanisms without verification. The first of these problems is the

one in which the objective is to minimize the *weighted sum* of all jobs completion times. Archer and Tardos [2] proved that, without verification, no truthful mechanism can achieve an approximation ratio better than $2/\sqrt{3}$. We instead show that there exists an *exact* truthful mechanism with verification. Interestingly enough, we prove this result by showing that *every* exact algorithm for this problem is roughly monotone. This indicates that roughly monotone algorithms are a "natural" class: exact solutions fulfill this requirement but not the monotonicity one [2]. We obtain analogous results for the problem of scheduling *selfish jobs* on related "unselfish" machines studied in [3]. In this case, we have *polynomial-time* mechanisms with verification that obtain an approximation smaller than the approximation factor of any mechanism without verification, even if the latter runs in *exponential* time. We describe a simple *polynomial-time* $(1+\varepsilon)$-approximation mechanisms with verification for this NP-hard problem, for every $\varepsilon > 0$. Notice that no mechanism without verification can attain an approximation factor better than $(1+\sqrt{17})/4$, no matter its running time [3].

Finally, in our view, one of the main contribution of this work is on the formal setting describing how verification is used. In particular, the mechanism cannot fine the agents and all it is allowed to do is to deny the payment to an agent that is caught lying. In order to make this "soft punishment" into a truthful mechanism we use *voluntary participation*: truthful agents do not run in a loss. Agents can report any type in a common domain and the type does not give any a priori information (unlike the mechanisms with "partial verification" considered in the classical game theory literature [11] – see Section 8). The model we propose is also slightly more general than the one in [9] where an agent that artificially delays the execution of the assigned job incurs a cost equal to the artificially delayed time. We discuss these issues more in detail in Section 8.

1.2.0.1 Road map. In the following section, we provide a formal setting for mechanism design optimization problems involving one-parameter agents. Besides giving the necessary notation and definitions, here we formalize the use of verification for one-parameter agents. In Section 3, we present a startup result for a special case called overbidding agents. We give our characterization in Section 4 and extend the positive result in Section 5 in order to guarantee computational efficiency for infinite domains. The application to the makespan minimization problem is given in Section 6. Section 7 contains results on other problems showing the power of verification for one-parameter agents. In Section 8 we conclude by providing connections and comparing to existing mechanism design techniques and results. We also present open problems and outline future research directions.

2 Formal setting

We are given a finite set 'Solutions' of feasible solutions and n selfish agents each of them assigning a monetary valuation to each solution. This monetary valuation depends on the *type* t_i of the corresponding agent i and is denoted by $v(x, t_i)$, with $x \in$ Solutions. For each agent i we have a *domain* D_i consisting of the range of all possible types t_i. The type t_i is known to agent i and each agent reports some (possibly different) type b_i in the domain D_i.

In the case of *one-parameter* agents, the valuations are all of the form

$$v(x, t_i) = -w(x, i) \cdot t_i,$$

where $w(x, i)$ and t_i are non-negative real numbers. In particular, $w(x, i)$ is the *work* that solution x assigns to agent i.

An algorithm A receives in input the *reported types* $b = (b_1, \ldots, b_n)$ and returns a feasible solution $A(b)$. We sometimes refer to the reported types as the agents' *bids*. Notice that an agent might have an incentive in misreporting her type if this leads A to return a solution which is better for that agent. For instance, agent i is better off if the work assigned to her reduces when misreporting her type to b_i (given the types reported by the other agents). If agent i receives a payment r and a solution x is chosen, then her *utility* is equal to

$$r + v(x, t_i) = r - w(x, i) \cdot t_i.$$

A *mechanism* is a pair (A, P) where A is an algorithm as above and $P = (P_1, \ldots, P_n)$ is a vector of payment functions: Given the vector b as above, the mechanism computes a feasible solution $A(b)$ and, for each agent i, a payment $P_i(b)$. For ease of notation, we denote $w(A(b), i)$ simply as $w_i^A(b)$. Mechanisms *without verification* provide each agent a payment $P_i(b)$ and thus the corresponding utility is

$$P_i(b) + v(A(b), t_i) = P_i(b) - w_i^A(b) \cdot t_i.$$

Let b_{-i} denote the vector of types reported by all agents but i, that is,

$$b_{-i} \stackrel{\text{def}}{=} (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n).$$

Each element in the vector b_{-i} is taken from the domain of the corresponding agent, that is, b_{-i} belongs to the set $D_{-i} \stackrel{\text{def}}{=} D_1 \times \cdots \times D_{i-1} \times D_{i+1} \times \cdots \times D_n$. We let (γ, b_{-i}) be the vector $(b_1, \ldots, b_{i-1}, \gamma, b_{i+1}, \ldots, b_n)$. We say that (A, P) is a *truthful mechanism* if truthtelling is a dominant strategy for all agents. That is, for every b and t as above, the following holds:

$$P_i(t_i, b_{-i}) - w_i^A(t_i, b_{-i}) \cdot t_i \geq P_i(b) - w_i^A(b) \cdot t_i.$$

In a *mechanism with verification* for one parameter agents, agent i receives the payment $P_i(b)$ if and only if agent i is *not caught lying*, that is, the case $w_i^A(b) \cdot b_i \geq w_i^A(b) \cdot t_i$. Otherwise, for $w_i^A(b) \cdot b_i < w_i^A(b) \cdot t_i$, agent i is *caught lying* and she does not receive any payment. So, the payment received by agent i in a mechanism with verification (A, P) is a function of her own type t_i and of the reported types b:

$$P_i^{ver}(b|t_i) \stackrel{\text{def}}{=} \begin{cases} P_i(b) \text{ if } w_i^A(b) \cdot b_i \geq w_i^A(b) \cdot t_i \quad \text{(not caught lying)}, \\ 0 \qquad \text{if } w_i^A(b) \cdot b_i < w_i^A(b) \cdot t_i \quad \text{(caught lying)}. \end{cases}$$

The definition of truthful mechanism extends naturally to mechanisms with verification. In particular, we say that (A, P) *is a truthful mechanism with verification for one-parameter agents* if the inequality

$$P_i^{ver}(t_i, b_{-i}|t_i) - w_i^A(t_i, b_{-i}) \cdot t_i \geq P_i^{ver}(b|t_i) - w_i^A(b) \cdot t_i$$

holds for all i, for all $t_i, b_i \in D_i$ and for all $b_{-i} \in D_{-i}$. Observe that, since $w_i^A(b) \geq 0$ for all b, the condition "$w_i^A(b) \cdot b_i \geq w_i^A(b) \cdot t_i$" is equivalent to "$w_i^A(b) = 0$ or $b_i \geq t_i$". By simple substitution we get an equivalent condition for being truthful with verification:

Theorem 1 *A pair (A, P) is a truthful mechanism with verification for one-parameter agents if and only if for all i, for all $t_i, b_i \in D_i$ and for all $b_{-i} \in D_{-i}$, it holds that*

$$P_i(t_i, b_{-i}) - w_i^A(t_i, b_{-i}) \cdot t_i \geq P_i(b) - w_i^A(b) \cdot t_i \qquad (\textit{not caught lying})$$

for $w_i^A(b) = 0$ or $b_i \geq t_i$, and

$$P_i(t_i, b_{-i}) - w_i^A(t_i, b_{-i}) \cdot t_i \geq -w_i^A(b) \cdot t_i \qquad (\textit{caught lying})$$

for $w_i^A(b) > 0$ and $b_i < t_i$.

A mechanism (A, P) satisfies the *voluntary participation* if the utility of a truthtelling agent is always nonnegative. That is, $P_i(t_i, b_{-i}) \geq w_i^A(t_i, b_{-i}) \cdot t_i$, for all i, for all $b_{-i} \in D_{-i}$, and for all $t_i \in D_i$. We say that a mechanism *uses* an algorithm A if the mechanism is a pair (A, P), for some P as above. An algorithm A *admits* a truthful mechanism (with verification) if there exists a truthful mechanism (with verification) that uses algorithm A.

We consider mechanisms for *optimization problems* involving one-parameter agents meaning that every problem instance is a pair (t, π), where t is the *private part* and π is the *public part* of the instance. The private part t is the vector $t = (t_1, \ldots, t_n)$ of the true agents' types. The set of feasible solutions of an instance (t, π) depends uniquely on its public part π, and it is denoted by Solutions(π). The *measure* of a solution $x \in$ Solutions(π) with

respect to the instance (t, π) depends on *both* t and π and it is denoted by $\text{Measure}(x, t, \pi)$. For *minimization* problems, the goal is to find a feasible solution for a given instance which minimizes the corresponding measure. An *exact algorithm* for a minimization problem, on input an instance (t, π), returns a feasible x such that $\text{Measure}(x, t, \pi) = \min_{y \in \text{Solutions}(\pi)} \text{Measure}(y, t, \pi)$. (For maximization problems, the condition is the same but with 'max' in place of 'min'.) A *c–approximation algorithm* for a minimization problem, on input (t, π), returns a feasible solution x such that $\text{Measure}(x, t, \pi) \leq c \cdot \min_{y \in \text{Solutions}(\pi)} \text{Measure}(y, t, \pi)$. Such an x is a *c–approximate solution* for (t, π). (For maximization problems, the condition is the same but with '$\leq$' and 'c' replaced by '$\geq$' and '$1/c$', respectively.) Following the standard definitions used in the literature, an optimization problem is a quadruple

$$(\text{Instances}, \text{Measure}, \text{Solutions}, \text{Goal}),$$

where Goal ='min' for minimization problems, and Goal ='max' for maximization problems. An optimization problem involving one-parameter agents is an optimization problem in which Instances $= D \times$ Public, where Public contains all possible public parts π of an instance and $D \stackrel{\text{def}}{=} D_1 \times \cdots \times D_n$ is the domain of all one-parameter agents. Hence, if (t, π) is an instance of the problem, then every (b, π) with $b \in D$ is also an instance of the problem. A mechanism for an optimization problem receives in input an instance (b, π), where $b = (b_1, \ldots, b_n)$ is a vector of types reported by the agents whose types are $(t_1, \ldots, t_n)$. So, the mechanism computes a feasible solution $A(b, \pi) \in \text{Solutions}(\pi)$ and, for each agent i, a payment $P_i(b, \pi)$. An optimization problem *admits a truthful mechanism* if, for every $\pi \in$ Public, there exists a truthful mechanism $(A^{(\pi)}, P^{(\pi)})$ such that $A^{(\pi)}(b) \in \text{Solutions}(\pi)$ for all $b \in D$. In this case, we say that (A, P) is a truthful mechanism for this optimization problem if $A(b, \pi) = A^{(\pi)}(b)$ and $P_i(b, \pi) = P_i^{(\pi)}(b)$, for all $b \in D$ and for all i. Such a mechanism satisfies the voluntary participation if all mechanisms $(A^{(\pi)}, P^{(\pi)})$ satisfy the voluntary participation. The mechanism (A, P) is a c–approximation mechanism if A is a c–approximation algorithm. Algorithm A is a *polynomial-time* algorithm if its running time is polynomial in the input (b, π), for every $b \in D$ and $\pi \in$ Public. Similarly, the payment functions $P = (P_1, \ldots, P_n)$ are polynomial-time if each P_i is computable in time polynomial in the size of (b, π), for $b \in D$ and $\pi \in$ Public. A mechanism (A, P) for an optimization problem is *polynomial-time* if *both* A and P are polynomial-time. I.e., the mechanism returns the solution and the payments in time polynomial in the size of the input. For *finite domains* D, each b requires $O(n \log |D|)$ bits, and thus polynomial running time means time polynomial in $O(n \log |D|)$ and in the size of π. For *infinite domains*, we allow agents to specify bids of arbitrary bit-length (i.e., rational numbers of arbitrary precision). In this case, the size of b (in number of bits using the standard encoding) is $\Omega(\max\{n, \log(b_{\max}/b_{\min})\})$, where $b_{\max}$ and $b_{\min}$ are the largest and the smallest element in b. So, polynomial running time is meant

as polynomial in the size of b and in the size of π.

Remark 2 *Throughout the paper we specify properties/conditions/definitions for algorithms A taking in input a bid vector b. We extend all of such properties/conditions/definitions to algorithms for optimization problems in the natural way. That is, the corresponding property/condition/definition needs to hold for* every *fixed $\pi \in$ Public. We also extend the notation $w_i^A(\cdot)$ by letting $w_i^A(b, \pi) \stackrel{\text{def}}{=} w(A(b, \pi), i)$. Again, all properties/conditions/definitions specified for $w_i^A(b)$ extend in the natural way by requiring them to hold for every fixed π with respect to $w_i^{A^{(\pi)}}(b) \stackrel{\text{def}}{=} w_i^A(b, \pi)$.*

3 Truthful mechanisms for overbidding agents

In this section we consider *overbidding* agents, that is, the scenario in which no agent can report a type b_i which is smaller than her own type t_i. A truthful mechanism for overbidding agents guarantees that, if every agent can only report $b_i \geq t_i$, with $b_i \in D_i$, then the utility of each agent i is maximized for $b_i = t_i$. As we will see below, these mechanisms constitute the "building block" for truthful mechanisms with verification. Overbidding agents have been studied by Singh and Wittman [11] where, among other results, the authors proved the next theorem below. We give here a proof since the payments will be used for obtaining mechanisms with verification and, to this aim, we will define payments satisfying a condition which is slightly stronger than truthfulness (see Remark 4 below).

Theorem 3 *Every algorithm admits a truthful mechanism for overbidding agents in the case of finite domains. The obtained mechanism also satisfies the voluntary participation condition.*

Proof. We show that, for any algorithm A, there exist P such that (A, P) is a truthful mechanism for overbidding agents in the case in which every D_i is finite. In order to guarantee that (A, P) is a truthful mechanism for overbidding agent it suffices to guarantee that, for every $b_{-i} \in D_{-i}$, and for every $b_i, t_i \in D_i$, with $b_i > t_i$, the following inequality holds:

$$P_i(t_i, b_{-i}) - w_i^A(t_i, b_{-i}) \cdot t_i \geq P_i(b) - w_i^A(b) \cdot t_i. \tag{1}$$

Let $D_i = \{d_i^1, \ldots, d_i^\ell\}$, where ℓ is the cardinality of D_i. Without loss of generality we consider these elements in increasing order, that is,

$$d_i^1 < \cdots < d_i^\ell.$$

First, we define the payment associated to the largest element d_i^ℓ in D_i:

$$P_i(d_i^\ell, b_{-i}) \stackrel{\text{def}}{=} w_i^A(d_i^\ell, b_{-i}) \cdot d_i^\ell. \tag{2}$$

Then, starting from this value, we define the payments for all other elements as follows:

$$P_i(d_i^j, b_{-i}) \stackrel{\text{def}}{=} w_i^A(d_i^j, b_{-i}) \cdot d_i^j + \max_{k>j} \left\{ P_i(d_i^k, b_{-i}) \right\}. \tag{3}$$

Observe that the value of $P_i(d_i^j, b_{-i})$ depends only on the pre-computed values

$$P_i(d_i^{j+1}, b_{-i}), \ldots, P_i(d_i^\ell, b_{-i}).$$

Now observe that, for every $t_i, b_i \in D_i$, with $b_i > t_i$, we have $t_i = d_i^j$ and $b_i = d_i^k$, for some $1 \leq j < k \leq \ell$. From (3) and from $w_i^A(d_i^j, b_{-i}) \cdot d_i^j \geq 0$, we conclude that (1) holds. That is, the mechanism (A, P) is truthful for overbidding agents. Finally, the voluntary participation holds because of (2–3) and because payments are nonnegative. □

Remark 4 *We stress that the mechanism of the above theorem guarantees a stronger version of truthfulness for overbidding agents. In particular, because of (3), we have that for every t_i and for every b such that $b_i > t_i$, it holds that $P_i(t_i, b_{-i}) - w_i^A(t_i, b_{-i}) \cdot t_i \geq P_i(b)$. Since $w_i^A(b) \cdot t_i \geq 0$, this condition implies truthfulness for overbidding agents, that is (1).*

4 A characterization

In this section, we *characterize* the class of truthful mechanisms with verification for one-parameter agents. As we will see, this result shows that mechanisms with verification require a much less stringent condition on the underlying algorithm, as compared to mechanisms *without* verification [8,2]. Another interesting consequence of this characterization is that designing truthful mechanisms with verification is "harder" than designing truthful mechanisms for overbidding agents (the latter type do not impose any restriction on the algorithm to be used – see Theorem 3). In Section 4.1, we provide a necessary condition. In Section 4.2, we show that this condition suffices for every finite domain and, more in general, if one is able to obtain a truthful mechanism for overbidding agents which also satisfies the voluntary participation condition (see Theorem 7). This gives a characterization of truthful mechanisms with verification for finite domains (see Corollary 8).

4.1 A necessary condition...

In this section we provide a *necessary* condition for obtaining truthful mechanisms with verification. In particular, we will show that, if (A, P) is a truthful mechanism with verification, then algorithm A must satisfy the following:

Definition 5 (roughly monotone algorithm) *An algorithm A is* roughly monotone *if, for all $b \in D$ and for all i, the following holds. If $w_i^A(b) = 0$ then $w_i^A(b'_i, b_{-i}) = 0$ for all $b'_i > b_i$ with $b'_i \in D_i$.*

For scheduling problems, roughly monotone algorithms satisfy the following natural condition. If the algorithm does not use machine i, then the algorithm does not use machine i if this machine becomes slower (provided the speeds of the other machines do not change).

Theorem 6 *Every truthful mechanism with verification must use a roughly monotone algorithm.*

Proof. We show that, if (A, P) is a truthful mechanism with verification, then A is roughly monotone. We proceed by way of contradiction and assume that algorithm A is not roughly monotone. Therefore, there exists a $d \in D$ and an i which violate the condition in Definition 5. That is, $w_i^A(d) = 0$ and $w_i^A(d'_i, d_{-i}) > 0$, for some $d'_i > d_i$, with $d'_i \in D_i$.

Since (A, P) is a truthful mechanism with verification, we can apply Theorem 1 with $t_i = d_i$, $b_i = d'_i$, and $b_{-i} = d_{-i}$. By definition, we have that $b_i > t_i$ and thus Theorem 1 (not caught lying) implies

$$P_i(d) - w_i^A(d) \cdot d_i \geq P_i(d'_i, d_{-i}) - w_i^A(d'_i, d_{-i}) \cdot d_i. \tag{4}$$

Similarly, for $t_i = d'_i$, $b_i = d_i$, and $b_{-i} = d_{-i}$, we have $b = d$ and thus $w_i^A(b) = 0$. Once again, Theorem 1 (not caught lying) implies

$$P_i(d'_i, d_{-i}) - w_i^A(d'_i, d_{-i}) \cdot d'_i \geq P_i(d) - w_i^A(d) \cdot d'_i. \tag{5}$$

Since $w_i(b) = w_i(d) = 0$, by summing up (4) with (5) we get

$$-w_i^A(d'_i, d_{-i}) \cdot d'_i \geq -w_i^A(d'_i, d_{-i}) \cdot d_i.$$

Since $w_i^A(d'_i, d_{-i}) > 0$ we have $d'_i \leq d_i$, thus contradicting the hypothesis $d'_i > d_i$. □

In this section we show that the necessary condition given in the previous section (i.e., roughly monotone algorithms) suffices for obtaining truthful mechanisms with verification, provided that we can guarantee the voluntary participation condition. Intuitively speaking, for roughly monotone algorithms, the construction of truthful mechanisms with verification "reduces" to the construction of a truthful mechanism for overbidding agents satisfying the voluntary participation. We stress that we do not make any assumption on the domain.

Theorem 7 *Every roughly monotone algorithm that admits a truthful mechanism for overbidding agents satisfying the voluntary participation also admits a truthful mechanism with verification. The latter mechanism satisfies the voluntary participation as well.*

Proof. Let A be roughly monotone and let (A, P) be a truthful mechanism for overbidding agents that also satisfies the voluntary participation (this mechanism exists because of Theorem 3). We first modify the payment function so that only the agents that receive a strictly positive work are assigned some payment. Consider the following payments:

$$P'_i(b) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } w_i^A(b) = 0, \\ P_i(b) & \text{otherwise.} \end{cases} \tag{6}$$

We first show that, since A is roughly monotone, then the new payments specify a mechanism (A, P') which is also truthful for overbidding agents and satisfies the voluntary participation. That is, for every i, b_{-i}, t_i, and $b_i > t_i$, the following inequality holds:

$$P'_i(t_i, b_{-i}) - w_i^A(t_i, b_{-i}) \cdot t_i \geq P'_i(b) - w_i^A(b) \cdot t_i. \tag{7}$$

If $w_i^A(t_i, b_{-i}) = 0$, then the fact that A is roughly monotone implies $w_i^A(b) = 0$. This and (6) imply that $P'_i(t_i, b_{-i}) = 0 = P'_i(b)$, and thus the above inequality holds. If $w_i^A(t_i, b_{-i}) > 0$ and $w_i^A(b) > 0$, then we simply observe that $P'_i(t_i, b_{-i}) = P_i(t_i, b_{-i})$ and $P'_i(b) = P_i(b)$. In this case, (7) follows from the fact that (A, P) is truthful for overbidding agents. Finally, we consider the case $w_i^A(t_i, b_{-i}) > 0$ and $w_i^A(b) = 0$. Since (A, P) satisfies the voluntary participation, we have $P_i(t_i, b_{-i}) \geq w_i^A(t_i, b_{-i}) \cdot t_i$. Moreover, from (6), we have $P'_i(t_i, b_{-i}) = P_i(t_i, b_{-i})$ and $P'_i(b) = 0$. So, the right hand side of (7) is equal to 0, and the inequality follows from $P_i(t_i, b_{-i}) \geq w_i^A(t_i, b_{-i}) \cdot t_i$. The voluntary participation of (A, P') follows from the observation that the only case in which $P'_i(t_i, b_{-i}) \neq P_i(t_i, b_{-i})$ is for $w_i^A(t_i, b_{-i}) = 0$, in which case we have

$P'_i(t_i, b_{-i}) = 0 = w_i^A(t_i, b_{-i}) \cdot t_i$. Otherwise, for $w_i^A(t_i, b_{-i}) > 0$, the voluntary participation of (A, P) implies $P'_i(t_i, b_{-i}) = P_i(t_i, b_{-i}) \geq w_i^A(t_i, b_{-i}) \cdot t_i$.

We next show that (A, P') is a truthful mechanism with verification. We distinguish the two cases in Theorem 1:

i **is not caught lying.** For $b_i \geq t_i$, since (A, P') is truthful for overbidding agents, we have

$$P'_i(t_i, b_{-i}) - w_i^A(t_i, b_{-i}) \cdot t_i \geq P'_i(b) - w_i^A(b) \cdot t_i.$$

For $b_i < t_i$, it must be $w_i^A(b) = 0$, since otherwise i is caught lying. Since A is roughly monotone, we have $w_i^A(t_i, b_{-i}) = 0$. From (6) we have $P'_i(b) = P'_i(t_i, b_{-i}) = 0$, thus implying that the above inequality holds also in this case.

i **is caught lying.** In this case we have $w_i^A(b) > 0$ and $b_i < t_i$. We need to show that the inequality in Theorem 1(caught lying) holds. Since (A, P') satisfies the voluntary participation, and since $w_i^A(b) \geq 0$ for all b, we have

$$P_i(t_i, b_{-i}) - w_i^A(t_i, b_{-i}) \cdot t_i \geq 0 \geq -w_i^A(b).$$

From Theorem 1 we conclude that (A, P') is a truthful mechanism with verification.

Finally, (A, P') is a mechanism with verification satisfying the voluntary participation. Indeed, we simply observe that, on input (t_i, b_{-i}), the mechanism with verification (A, P') awards agent i a payment $P_i^{ver}(t_i, b_{-i}|t_i) = P'_i(t_i, b_{-i}) \geq w_i(t_i, b_{-i}) \cdot t_i$, where the inequality has been proved above when considering the pair (A, P') as a mechanism for overbidding agents (satisfying the voluntary participation). □

From Theorems 3, 6 and 7 we obtain the following characterization of truthful mechanisms with verification in the case of finite domains:

Corollary 8 *In the case of finite domains, an algorithm admits a truthful mechanism with verification if and only if it is roughly monotone.*

We note that the "if part" of the above theorem does *not* require finite domains in general. Indeed, in Section 5.2 we will relax this assumption and provide mechanisms for infinite domains.

5 Computational efficiency and approximation

We start observing that the payments of Theorem 7 (see Equation 6) are essentially the payments of the mechanism for overbidding agents. Hence, we can extend the result of Theorem 7 to guarantee polynomial running time:

Corollary 9 *Every roughly monotone algorithm that admits a polynomial-time truthful mechanism for overbidding agents satisfying the voluntary participation also admits a polynomial-time truthful mechanism with verification. The latter mechanism satisfies the voluntary participation as well.*

Notice that the above result is very general since we make no assumption on the agents domains. In the sequel we will derive computationally efficient mechanisms with verification by constructing computationally efficient truthful mechanisms for overbidding agents. Observe that Theorem 3 says that, for finite domains, there always exists a truthful mechanism for overbidding agents satisfying the voluntary participation. However, a direct use of this theorem does not yield a polynomial-time mechanism. In particular, the time to compute each payment function $P_i(\cdot)$ is $\Omega(|D_i|)$, which in general is *not* polynomial in the size of the input (b, π).

In Section 5.1, we provide a technique for computing the payments efficiently for finite domains. The idea is to round the agents' bids, while preserving approximability and truthfulness. In Section 5.2, we extend this approach to *infinite domains* by considering algorithms which ignore bids that are "too large".

5.1 First, finite domains efficiently...

In this section we describe a class of optimization problems for which rounding the input bids has only a small impact on the quality of the solution produced by the algorithm. In this case, for any algorithm A we construct an algorithm $\hat{A}$ that simply works on the rounded bids. In this way, the payments computation needs to consider only a *subset* of the domain (i.e., the possible rounded bids) and can be thus performed in *polynomial time*. In doing so, we only lose an "arbitrary small" factor in the approximation guarantee of the original algorithm A.

Let us now proceed formally.

Definition 10 *An optimization problem is* smooth *if, for every $\varepsilon > 0$, there exists $\gamma = \gamma(\varepsilon)$ such that, for every $t', t'' \in D$ with $\max_i\{t'_i/t''_i, t''_i/t'_i\} \leq \gamma$, the following holds. Every c–approximate solution for the instance (t', π) is also a*

$c(1+\varepsilon)$–*approximate solution for the instance* (t'', π)*, for every* $\pi \in$ Public.

Now a natural approach suggests itself. We round each agent reported type b_i to the closest power of $\gamma = \gamma(\varepsilon)$. In particular, for any c–approximation algorithm A, the algorithm $\hat{A}$ which consists of running A on the rounded types is a $c(1+\varepsilon)$–approximation algorithm. Moreover, the number of possible values for each agent type is *logarithmic* in the size of the domain D_i. Then, a mechanism $(\hat{A}, P)$ which is truthful for these rounded types is also computationally efficient.

Theorem 11 *In the case of finite domains, every polynomial-time roughly monotone c–approximation algorithm for a smooth problem admits a polynomial-time* $c(1+\varepsilon)$*–approximation truthful mechanism with verification, for every* $\varepsilon > 0$*. In particular, both the solution and the payments can be computed in time polynomial in the size of the input. Moreover, the mechanism satisfies the voluntary participation.*

Proof. Let A be any algorithm satisfying the hypothesis of the theorem. We start by rounding the bids up to the closest power of $\gamma = \gamma(\varepsilon)$ and give these rounded values in input to A. Since the problem is smooth, this defines a polynomial-time roughly monotone $c(1+\varepsilon)$–approximation algorithm $\hat{A}$. From Corollary 9, it suffices to show that algorithm $\hat{A}$ admits a polynomial-time $(1+\varepsilon)$–approximation mechanism for overbidding agents and that the mechanism satisfies the voluntary participation. We prove the latter result in two steps. We first define a mechanism for a domain containing all possible powers of γ, and then extend the mechanism to the original domain.

For every x, we let $\mathrm{up}(x)$ be the value obtained by rounding x up to the smallest integer power of γ, that is,

$$\mathrm{up}(x) \stackrel{\text{def}}{=} \min\{\gamma^p |\ \gamma^p \geq x, \quad p \in \mathbf{Z}\}.$$

We extend the function $\mathrm{up}(\cdot)$ to sets and to vectors in the natural way. We first define a payment function P such that (A, P) is a truthful mechanism for overbidding agents with domain $\mathrm{up}(D)$. To this end, for every i, we let

$$\hat{D}_i \stackrel{\text{def}}{=} \mathrm{up}(D_i)$$

and $\hat{D} \stackrel{\text{def}}{=} \hat{D}_1 \times \cdots \times \hat{D}_n = \mathrm{up}(D)$. Let P be the payment function of Theorem 3 relatively to the domain $\hat{D}$ only. Recall that the mechanism (A, P) satisfies the following condition (see Remark 4):

$$P_i(\hat{t}_i, \hat{b}_{-i}) - w_i^A(\hat{t}_i, \hat{b}_{-i}) \cdot \hat{t}_i \geq P_i(\hat{b}), \tag{8}$$

for all $\hat{b}_{-i} \in \hat{D}_{-i}$ and $\hat{t}_i, \hat{b}_i \in \hat{D}_i$, with $\hat{b}_i > \hat{t}_i$. We now prove that the natural

extension

$$\hat{A}(\cdot) \stackrel{\text{def}}{=} A(\text{up}(\cdot)) \quad \text{and} \quad \hat{P}_i(\cdot) \stackrel{\text{def}}{=} P_i(\text{up}(\cdot)) \tag{9}$$

of the mechanism (A, P) is truthful for overbidding agents with respect to the *original domain* D. In particular, for any i, $b_{-i} \in D_{-i}$, and any $t_i, b_i \in D_i$, with $b_i > t_i$, we let $\hat{t}_i = \text{up}(t_i)$, $\hat{b}_i = \text{up}(b_i)$, $\hat{b}_{-i} = \text{up}(b_{-i})$. Observe that $\hat{b}_i \geq \hat{t}_i$ and that we only need to consider the case $\hat{b}_i > \hat{t}_i$, since for $\hat{b}_i = \hat{t}_i$ the mechanism outputs the some solution and the same payment. Then, observe that

$$\begin{aligned}
\hat{P}_i(t_i, b_{-i}) - w_i^{\hat{A}}(t_i, b_{-i}) \cdot t_i = & \quad \text{(from (9))} && (10)\\
P_i(\hat{t}_i, \hat{b}_{-i}) - w_i^{A}(\hat{t}_i, \hat{b}_{-i}) \cdot t_i \geq & \quad \text{(from } \hat{t}_i \geq t_i \text{ and } w_i^A(\cdot) \geq 0) && (11)\\
P_i(\hat{t}_i, \hat{b}_{-i}) - w_i^{A}(\hat{t}_i, \hat{b}_{-i}) \cdot \hat{t}_i \geq & \quad \text{(from (8) and from } \hat{b} = \text{up}(b)) && (12)\\
P_i(\hat{b}) = & \quad \text{(from (9))} && (13)\\
\hat{P}_i(b) \geq & \quad \text{(from } w_i^{\hat{A}}(\cdot) \geq 0 \text{ and } t_i \geq 0) && (14)\\
\hat{P}_i(b) - w_i^{\hat{A}}(b) \cdot t_i \ . & && (15)
\end{aligned}$$

Also notice that, since the payments P of Theorem 3 are always nonnegative, then the voluntary participation follows from (10–13). □

5.2 ...then, infinite domains efficiently too

In this section, we provide constructions of computationally efficient mechanisms for *infinite domains*, that is, we assume bids to be any *strictly positive* real number. Notice that our construction applies to *any* domain in which all bids are strictly positive reals. In particular, the size of the domain can be infinite.

The idea is to consider algorithms for which "too high" bids can be ignored in the sense that the corresponding agent will be assigned no work. For instance, in the $\text{Q}||\text{C}_{\max}$ problem, optimal solutions do not allocate any job to a machine (agent) if its speed is "much worse" than the speed of any other machine.

Definition 12 (bounded algorithm) *A roughly monotone algorithm A is* bounded *if for all b and for all i, there exists $B_i = B_i(b_{-i})$ such that $w_i^A(b'_i, b_{-i}) = 0$, for all $b'_i \geq B_i$. A bounded algorithm is* polynomially-bounded *if B_i is polynomial in n and in $\max_{j,k}\{b_j/b_k\}$, for all b and for all i.*

Intuitively speaking, bounded algorithms allow us to "reduce" the case of infinite domains to the case of finite domains studied in the previous section. First, we "discretize" the domain by considering rounded bids (as in

Section 5.1). Then, we build the payments by considering the "semi-finite" domain consisting of the rounded bids up to B_i. The fact that the algorithm is polynomially-bounded will imply that these payments can be computed in polynomial time, if the algorithm runs in polynomial time. Formally, we have the following result extending Theorem 11:

Theorem 13 *Every polynomial-time polynomially-bounded c–approximation algorithm for a smooth problem admits a polynomial-time $c(1+\varepsilon)$–approximation truthful mechanism with verification, for every $\varepsilon > 0$. In particular, both the solution and the payments can be computed in time polynomial in the size of the input. Moreover, the mechanism satisfies the voluntary participation.*

PROOF. The proof essentially combines the ideas used in the proofs of Theorems 3 and 11. We start by rounding the bids to the closest power of $\gamma = \gamma(\varepsilon)$ in input a c–approximation algorithm A, and obtain in this way a $c(1+\varepsilon)$–approximation algorithm $\hat{A}$. Then, we build our payment functions upon a "discretized" domain $\hat{D}$ consisting of all rounded bids. As we will see, these payment functions P_i can be extended in the natural way and lead to a truthful mechanism for the infinite domain.

For every $x > 0$, we let $\mathrm{up}(x)$ be the value obtained by rounding x up to the smallest integer power of $\gamma = \gamma(\varepsilon)$, that is,

$$\mathrm{up}(x) \stackrel{\text{def}}{=} \min\{\gamma^p |\ \gamma^p \geq x, \quad p \in \mathbf{Z}\}.$$

Definition 10 implies that $\hat{A} \stackrel{\text{def}}{=} A(\mathrm{up}(\cdot))$ is a polynomial-time $c(1+\varepsilon)$–approximation algorithm. From Corollary 9 it is sufficient to construct a polynomial-time mechanism $(\hat{A}, \hat{P})$ which is truthful for overbidding agents satisfying the voluntary participation.

We first consider the rounded domain $\hat{D} \stackrel{\text{def}}{=} \mathrm{up}(D)$. Let $\hat{D}_i \stackrel{\text{def}}{=} \mathrm{up}(D_i)$ and $\hat{B}_i \stackrel{\text{def}}{=} B_i(\hat{b}_{-i})$. In order to guarantee truthfulness w.r.t. the infinite domain, we require a condition which is slightly stronger than truthfulness for overbidding agents (essentially the same as in Remark 4). The condition is that, for every $\hat{b}_{-i} \in \hat{D}_{-i} = \mathrm{up}(D_{-i})$, for every $\hat{t}_i, \hat{b}_i \in \hat{D}_i$ with $\hat{b}_i > \hat{t}_i$, the following inequality holds:

$$P_i(\hat{t}_i, \hat{b}_{-i}) - w_i^A(\hat{t}_i, \hat{b}_{-i}) \cdot \hat{t}_i \geq P_i(\hat{b}). \tag{16}$$

Consider an arbitrary $\hat{b}_{-i} \in \hat{D}_{-i}$ and $\hat{d}_i \in \hat{D}_i$ and let us define the corresponding payment $P_i(\hat{d}_i, \hat{b}_{-i})$ as follows. If $\hat{d}_i \geq \hat{B}_i = B_i(\hat{b}_{-i})$, then we let

$$P_i(\hat{d}_i, \hat{b}_{-i}) \stackrel{\text{def}}{=} w_i^A(\hat{B}_i, \hat{b}_{-i}) \cdot \hat{B}_i = 0, \tag{17}$$

where the second inequality follows from the definition of $\hat{B}_i$. Otherwise, for

every $\hat{d}_i < \hat{B}_i$, we define the payment recursively:

$$P_i(\hat{d}_i, \hat{b}_{-i}) \stackrel{\text{def}}{=} w_i^A(\hat{d}_i, \hat{b}_{-i}) \cdot \hat{d}_i + \max_{\hat{b}_i \in \hat{D}_i, \hat{d}_i < \hat{b}_i \leq \hat{B}_i} \left\{ P_i(\hat{b}_i, \hat{b}_{-i}) \right\}. \tag{18}$$

The crucial observation here is that, in the above equation, the payments for $\hat{d}_i$ depend only on the payments for $\hat{b}_i > \hat{d}_i$ and that these are *finitely many*. Indeed, since $d_i > 0$, we have that $\hat{d}_i = \gamma^p$, for some integer p. Hence, the set of all $\hat{b}_i > \hat{d}_i$ in $\hat{D}_i$ is finite and consists of the following elements (notice that $\gamma > 1$ because of Definition 10):

$$\gamma^{p+1} = \hat{d}_i \cdot \gamma < d_i \cdot \gamma^2 < \cdots < \hat{d}_i \cdot \gamma^\ell = \hat{B}_i$$

with $\gamma^\ell = \hat{B}_i / \hat{d}_i$. From (18) it follows that the time to compute the payment $P_i(\hat{d}_i, \hat{b}_{-i})$ is $O(T_A \cdot \ell^2)$, with T_A being the running time of algorithm A (below we will prove that this computation is polynomial-time). Now observe that (16) comes directly from (17–18) and from the fact that $w_i^A(\hat{d}_i, \hat{b}_{-i}) = 0$ for all $\hat{d}_i \geq \hat{B}_i$.

We now prove that the natural extension

$$\hat{A}(\cdot) \stackrel{\text{def}}{=} A(\text{up}(\cdot)) \quad \text{and} \quad \hat{P}_i(\cdot) \stackrel{\text{def}}{=} P_i(\text{up}(\cdot)) \tag{19}$$

of the mechanism (A, P) is truthful for overbidding agents with respect to the *infinite domain*. In particular, for any i, b_{-i}, and any t_i, b_i, with $b_i > t_i$, we consider $\hat{t}_i = \text{up}(t_i)$, $\hat{b}_i = \text{up}(b_i)$, and $\hat{b}_{-i} = \text{up}(b_{-i})$. Observe that we only need to consider the case $\hat{b}_i > \hat{t}_i$, since for $\hat{b}_i = \hat{t}_i$ the mechanism outputs the some outcome and the same payment. Then, observe that

$$\begin{aligned}
\hat{P}_i(t_i, b_{-i}) - w_i^{\hat{A}}(t_i, b_{-i}) \cdot t_i = & \quad \text{(from (19))} && (20)\\
P_i(\hat{t}_i, \hat{b}_{-i}) - w_i^A(\hat{t}_i, \hat{b}_{-i}) \cdot t_i \geq & \quad \text{(from } \hat{t}_i \geq t_i \text{ and } w_i^A(\cdot) \geq 0) && (21)\\
P_i(\hat{t}_i, \hat{b}_{-i}) - w_i^A(\hat{t}_i, \hat{b}_{-i}) \cdot \hat{t}_i \geq & \quad \text{(from (16) and from } \hat{b} = \text{up}(b)) && (22)\\
P_i(\hat{b}) = & \quad \text{(from (19))} && (23)\\
\hat{P}_i(b) \geq & \quad \text{(from } w_i^{\hat{A}}(\cdot) \geq 0 \text{ and } t_i \geq 0) && (24)\\
\hat{P}_i(b) - w_i^{\hat{A}}(b) \cdot t_i \ . & && (25)
\end{aligned}$$

From (17-18) it follows that the payments P are always nonnegative. Then, the voluntary participation follows from (20–23).

This completes the proof. □

Notice that we use polynomially-bounded algorithms to guarantee polynomial running time. However, if we do not consider computational issues, bounded

algorithms guarantee the existence of a truthful mechanism with verification for *infinite* domains:

Corollary 14 *Every smooth problem that admits an exact (respectively, c–approximation) bounded algorithm admits a $(1+\varepsilon)$–approximation (respectively, $c(1+\varepsilon)$–approximation) truthful mechanism with verification. The mechanism satisfies the voluntary participation.*

Throughout this section we have assumed that bids are strictly positive. We can extends all of our results by including the value '0' in the domain. In this case, it suffices that the algorithm satisfies the following condition: $w_i^A(b_i, b_{-i})$ is constant for all b_i in $[0, B_i')$, for some $B_i' = B_i'(b_{-i})$.

6 A $(1+\varepsilon)$–approximation mechanism for the makespan

In this section, we apply the constructions of computationally-efficient mechanisms given in Section 5.1 to the problem of scheduling jobs on (selfish) related machines. In particular, we present the first truthful polynomial-time $(1+\varepsilon)$–approximation mechanism for *any* number of selfish machines and for every $\varepsilon > 0$.

Theorem 15 *For every $\varepsilon > 0$, there exists a polynomial-time $(1+\varepsilon)$–approximation truthful mechanism with verification for $\mathrm{Q}||\mathrm{C}_{\max}$ when machine speeds are bounded by a constant.*[1] *Moreover, the mechanism satisfies the voluntary participation.*

In order to prove the theorem above, we modify the well-known polynomial-time approximation scheme (PTAS)[2] that Hochbaum and Shmoys [4] give for the case of *identical* speeds. This modification is necessary in order to consider machines with different speeds and to guarantee rough monotonicity. We stress that we need to restrict to speeds bounded by some constant *only* to guarantee polynomial running time of the modified PTAS. Theorem 13 (see also Definition 16 below) implies that every polynomial-time roughly monotone c–approximation algorithm for arbitrary speeds can be turned into a polynomial-time truthful mechanism with verification for arbitrary speeds with "roughly" the same approximation guarantee.

[1] We assume $s_i \in [1, R]$, with R being some constant value. By rescaling, we can easily extend the results to the case speeds are positive numbers in a range $[Min, Max]$ – see Remark 27 at page 29. Notice also that the lower bound $s_i \geq 1$ is necessary since allowing arbitrary small speeds makes the upper bound R meaningless.

[2] A polynomial-time approximation scheme is a family of algorithms such that, for every $\varepsilon > 0$, the family contains a polynomial-time $(1+\varepsilon)$-approximation algorithm for the corresponding optimization problem.

Let us first recall the problem formulation and observe that Theorem 13 can be indeed applied:

Definition 16 (Problem input and formulation for $Q||C_{max}$.) *The* public part π *of the input consists of a vector* $W = (W_1, \ldots, W_m)$ *of all jobs weights. The* private part *of the input consists of the vector* $t = (t_1, \ldots, t_n)$ *of machines processing times (equivalently, each machine* i *has a speed equal to* $s_i \stackrel{\text{def}}{=} 1/t_i$ *and its completion time, when assigned work* w_i*, is* $w_i/s_i = w_i \cdot t_i$*). The* feasible solutions *consists of all assignments of jobs to machines. For every feasible solution* x*, the work* $w(x, i)$ *assigned to agent* i *is the sum of the job weights that* x *assigns to machine* i*. The* optimization function *is the* makespan*, that is,* $\text{Measure}(x, t, W) = \max_i w(x, i) \cdot t_i$*. Clearly this optimization problem is smooth (see Definition 10) and every* c*-approximation algorithm for it must be polynomially-bounded (see Definition 12), for every constant* c*.*

In the description of the PTAS we follow a top-down approach and use the same nomenclature of Vazirani [13] describing the original PTAS in [4]. The modified PTAS is a family of algorithms $\textsc{Scan}_\varepsilon$ where:

(1) Algorithm $\textsc{Scan}_\varepsilon$ is a $(1+\varepsilon)$–approximation algorithm for $Q||C_{max}$ that uses another algorithm $\textsc{Oracle}_\delta$ as subroutine (δ is a suitable parameter depending on ε).
(2) Algorithm $\textsc{Oracle}_\delta$ receives from $\textsc{Scan}_\varepsilon$ a bound C and attempts to compute a feasible solution of cost at most $C \cdot (1+\delta)$. This algorithm uses another algorithm Exact as subroutine to compute an optimal assignment for a subset of "large" jobs and then "completes" this solution by assigning the remaining ("small") jobs greedily.
(3) Algorithm Exact uses a dynamic-programming approach to compute the optimal assignment of the "large" jobs. Its running time is polynomial if the different weights of the "large" jobs are a constant number.

In the sequel, we follow this approach and show the properties of each algorithm and how they can be derived from the corresponding subroutine. Ultimately, we will "reduce" the construction of the entire mechanism to the construction of a suitable algorithm Exact.

6.1 Algorithm $\textsc{Scan}_\varepsilon$

In this section we provide an overview of algorithm $\textsc{Scan}_\varepsilon$. We assume to have a family $\textsc{Oracle}_\delta$ of polynomial-time algorithms such that, for every $\delta > 0$, on input $C > 0$ and an instance (t, W), $\textsc{Oracle}_\delta$ returns either a scheduling of makespan at most $C \cdot (1+\delta)$ or fail in which case no assignment of makesapn

smaller than C exists. We will describe algorithm $\textsc{Oracle}_\delta$ in Section 6.2. Algorithm $\textsc{Scan}_\varepsilon$ searches for the smallest integer p for which $\textsc{Oracle}_\delta$ does not return fail on input $C = (1+\delta)^p$, where $\delta = \varepsilon/3$. (As we will see below, this choice of δ guarantees that $\textsc{Scan}_\varepsilon$ is a $(1+\varepsilon)$–approximation algorithm.) Figure 1 shows the pseudocode of algorithm $\textsc{Scan}_\varepsilon$.

```
Algorithm Scan_ε(t, W)
  δ ← ε/3;
  p ← 0;
  if Oracle_δ((1+δ)^p, t, W) = fail then
        while Oracle_δ((1+δ)^p, t, W) = fail do p = p + 1;
  else
        while Oracle_δ((1+δ)^p, t, W) ≠ fail do p = p − 1;
        p = p + 1;
  return Oracle_δ((1+δ)^p, t, W).
```

Fig. 1. Algorithm $\textsc{Scan}_\varepsilon$.

We next show that if the family Oracle enjoys the following three properties then $\textsc{Scan}_\varepsilon$ is $(1+\varepsilon)$–approximation roughly monotone algorithm (each of the three properties need to hold for every instance (t, W), for every $C > 0$, and for every $\delta > 0$):

(1) *Closeness.* If $\textsc{Oracle}_\delta(C, t, W) = \mathsf{fail}$ then $\mathrm{Opt}(t, W) \geq C$. If instead

$$\textsc{Oracle}_\delta(C, W, s) \neq \mathsf{fail}$$

then $\textsc{Oracle}_\delta$ returns a solution of makespan at most $C \cdot (1+\delta)$.

(2) *Stability.* If $x = \textsc{Oracle}_\delta(C, t, W) \neq \mathsf{fail}$ and $w(x, i) = 0$ then for all $b_i > t_i$ it holds that $\textsc{Oracle}_\delta(C, (b_i, t_{-i}), W) = \textsc{Oracle}_\delta(C, t, W)$.

(3) *Cost-Monotonicity.* If $x = \textsc{Oracle}_\delta(C, t, W) \neq \mathsf{fail}$ and $w(x, i) = 0$ then for all $C' < C$ such that $x' = \textsc{Oracle}_\delta(C', t, W) \neq \mathsf{fail}$, $w(x', i) = 0$.

We first show that the first of the three conditions above (Closeness) guarantees the desired approximation factor for $\textsc{Scan}_\varepsilon$. The proof follows closely the one given in [4].

Theorem 17 (essentially due to [4]) *If* Oracle *satisfies the Closeness condition, then* $\textsc{Scan}_\varepsilon$ *is a* $(1+\varepsilon)$*–approximation algorithm.*

Proof. Let h be the integer at which $\textsc{Scan}_\varepsilon$ stopped on input (t, W) and let x be the solution returned. Hence, $x = \textsc{Oracle}_\delta(C, t, W)$, for $C = (1+\delta)^h$, and $\textsc{Oracle}_\varepsilon((1+\delta)^{h-1}, t, W) = \mathsf{fail}$. The first part of the Closeness condition implies

$$\mathrm{Opt}(t, W) \geq (1+\delta)^{h-1}$$

while from the second part we obtain

$$\text{Measure}(x,t,W) \leq (1+\delta)^{h+1} \leq (1+\delta)^2 \text{Opt}(t,W) \leq (1+\varepsilon)\text{Opt}(t,W),$$

where the last inequality follows from $\delta = \varepsilon/3$. □

We next show that the last two conditions (Stability and Cost-Monotonicity) imply that algorithm $\textsc{Scan}_\varepsilon$ is roughly monotone.

Theorem 18 *If* Oracle *satisfies conditions Stability and Cost-Monotonicity then* $\textsc{Scan}_\varepsilon$ *is roughly monotone.*

Proof. Consider any b and $b' = (b'_i, b_{-i})$ with $b'_i > b_i$, and let W be any sequence of job weights. In order to show that $\textsc{Scan}_\varepsilon$ is roughly monotone, we show that $w(x,i) = 0$ implies $w(x',i) = 0$, where $x = \textsc{Scan}_\varepsilon(b,W)$ and $x' = \textsc{Scan}_\varepsilon(b',W)$. To this end, we consider the values $C, C' > 0$ such that

$$x = \textsc{Scan}_\varepsilon(b,W) = \textsc{Oracle}_\delta(C,b,W)$$

and

$$x' = \textsc{Scan}_\varepsilon(b',W) = \textsc{Oracle}_\delta(C',b',W).$$

We distinguish three cases:

($C' = C$.) In this case both x and x' are solutions computed by algorithm $\textsc{Oracle}_\delta$ with respect to the same bound C. Since the family Oracle satisfies condition Stability, $w(x,i) = 0$ implies $x = x'$, and thus $w(x',i) = 0$.

($C' < C$.) Since $w(x,i) = 0$, condition Stability implies

$$\textsc{Oracle}_\delta(C,b',W) = \textsc{Oracle}_\delta(C,b,W).$$

We can thus apply condition Cost-Monotonicity with $x = \textsc{Oracle}_\delta(C,b',W) \neq \mathsf{fail}$ and $x' = \textsc{Oracle}_\delta(C',b',W) \neq \mathsf{fail}$, thus implying that $w(x',i) = 0$.

($C' > C$.) We show that this case is not possible. In particular, we prove that

$$\textsc{Oracle}_\delta(C'',b',W) = \mathsf{fail}$$

for all $C'' < C'$, thus contradicting the hypothesis $x' = \textsc{Oracle}_\delta(C',b',W) \neq \mathsf{fail}$. Step 3 of $\textsc{Scan}_\varepsilon$ ensures that $\textsc{Oracle}_\delta(C'',b',W) = \mathsf{fail}$ for both $C'' = C' \cdot (1+\delta)^{-1}$ and $C'' = C' \cdot (1+\delta)^{-2}$. Condition Closeness implies that $\text{Opt}(b',W) \geq C'(1+\delta)^{-1}$. Hence, for $C'' \leq C' \cdot (1+\delta)^{-3}$, $\textsc{Oracle}_\delta(C'',b',W) = \mathsf{fail}$, since otherwise we would have a solution of makespan at most $C' \cdot (1+\delta)^{-2} < C' \cdot (1+\delta)^{-1} \leq \text{Opt}(b',W)$.

□

6.2 *Algorithm* ORACLE

We describe the family of algorithms ORACLE and prove that it satisfies the three conditions given in the previous section (Closeness, Stability, and Cost-Monotonicity). The proof is conditional to certain properties of algorithm EXACT, which we describe in Appendix A.

Algorithm ORACLE$_\delta$ receives in input a bound C and an instance (t, W). Then, it performs the following steps:

(1) *Initialization* (*Partition jobs into large and small ones*). If $C < W_{\max} \cdot t_{\min}$ then ORACLE$_\delta$ returns fail. Otherwise, ORACLE$_\delta$ partitions the jobs W into two sets: a set $L = \{L_1, \ldots, L_l\}$ of *large* jobs consisting of all the jobs of weight larger than $\delta \cdot C$, and a set $S = \{S_1, \ldots, S_{m-l}\}$ of *small* jobs consisting of all the jobs of weight at most $\delta \cdot C$.

(2) *Phase I* (*Scheduling large jobs optimally*). ORACLE$_\delta$ rounds the weight of each large job $L_h \in L$ down to L'_h computed as the maximum value $\delta \cdot C \cdot (1+\delta)^p$, with p integer, that is not greater than L_h (hence, $L'_h \leq L_h$). We denote by $L' = (L'_1, \ldots, L'_l)$ the sequence of the rounded weights. ORACLE$_\delta$ computes an optimal solution for the jobs L' using algorithm EXACT (described in Appendix A). If this solution has makespan greater than C (with respect to the rounded weights) then ORACLE$_\delta$ returns fail. Otherwise, we let $x^{(1)}$ be the assignment for the large jobs produced by EXACT and let ORACLE$_\delta$ continue with the next phase in order to schedule the small jobs.

(3) *Phase II* (*Scheduling small jobs greedily*). Small jobs are considered one by one and job S_i is assigned to the machine that minimizes its completion time with respect to the load assigned to it by $x^{(1)}$ and by the allocation of the previous small jobs. Ties are broken by selecting the machine with better speed and then the one with smaller index. We denote by $x^{(2)}$ the solution obtained at the end of this phase. If $x^{(2)}$ has makespan greater than $C \cdot (1+\delta)$ (with respect to the real weights of the jobs) then ORACLE$_\delta$ returns fail. Otherwise, ORACLE$_\delta$ continues with the adjustment phase.

(4) *Phase III* (*Adjustment*). In the final adjustment phase the algorithm partitions the machines into two sets, *slow machines* and *fast machines*, and then modifies the solution $x^{(2)}$ in order to guarantee that each of the $f(C, t, W)$ fast machines receives at least one job. The partition is computed in the following way. For every k, from 1 up to n, we check if it is possible to assign (exactly) one job to the k fastest machines without exceeding the bound $C \cdot (1+\delta)$. We assume, without loss of generality, that the number of jobs is at least the number of machines. Let $M^{(k)}$ denote the set of k fastest machines (ties are broken by selecting the machine with smaller index). Let $matching^{(k)}$ be the assignment in which the i^{th} smallest job goes to the i^{th} slowest machine in the set $M^{(k)}$ (only the

k smallest jobs are assigned, the k fastest machines receive exactly one job, and the other machines receive no jobs). We define $f(C, W, s)$ as the largest k such that $matching^{(k)}$ has makespan at most $C \cdot (1 + \delta)$, with respect to the vector t and the k smallest jobs of W only. The $f(C, t, W)$ fastest machines are the *fast machines* and the remaining $n - f(C, t, W)$ are the *slow machines*.

We obtain a solution $x^{(3)}$ as follows. Starting with solution $x^{(2)}$, we simply iterate the following step: Until there is a fast machine i with no work, we move the job that $matching^{(f(C,t,W))}$ assigns to i to this machine (notice that this may leave another fast machine with no work). Since every job is moved at most once, this process ends after at most m steps.

6.2.1 Proving the Closeness condition

We begin by proving the first of the three fundamental conditions that ORACLE must satisfy.

Lemma 19 ORACLE *satisfies the Closeness condition.*

PROOF. Let $\delta > 0$ and observe that ORACLE_δ returns fail because of one of the following three reasons:

(1) $C < W_{\max} \cdot t_{\min}$.
Since the completion time of just the largest job $W_{\max}$ on any machine is at least $W_{\max} \cdot t_{\min}$, we have $\text{Opt}(t, W) \geq W_{\max} \cdot t_{\min} > C$.

(2) The assignment of the large jobs (with respect to the rounded weights) has makespan greater than C.
Since EXACT computes an optimal solution of L' (the set of rounded large jobs), we have that

$$\text{Opt}(t, W) \geq \text{Opt}(t, L) \geq \text{Opt}(t, L^{'}) \geq C,$$

where the second inequality follows from $L'_h \leq L_h$, for each job h.

(3) In Phase II, a small job is assigned to a machine causing the completion time of that machine to exceed $C \cdot (1 + \delta)$.
Let S_j be the first small job that causes the makespan to exceed $C \cdot (1 + \delta)$ and let i be the machine that receives S_j. Then, before S_j is assigned to i, the completion time of i is at least $C \cdot (1 + \delta) - S_j \cdot t_i \geq C \cdot (1 + \delta) - S_j \geq C \cdot (1 + \delta) - C \cdot \delta = C$. (Recall that $t_i = 1/s_i$ and $s_i \in [1, R]$, thus implying $t_i \leq 1$.) Since i is the machine that minimizes the completion time it must be the case that all machines had completion time at least C. This implies that the optimal solution has makespan at least C, that is, $\text{Opt}(t, W) \geq C$.

Suppose now that $\textsc{Oracle}_\delta(C, t, W)$ does not return fail. Then the cost of $x^{(2)}$ computed at the end of Phase II is at most $C \cdot (1 + \delta)$. We have to show that the cost does not exceed $C \cdot (1 + \delta)$ after Phase III, that is, the makespan of $x^{(3)}$ is at most $C \cdot (1 + \delta)$. Simply observe that each of the machines for which an adjustment step has been performed receives (only) the same job it was assigned by $matching^{(f(C,t,W))}$. By definition of $f(C, t, W)$, the corresponding completion time is at most $C \cdot (1 + \delta)$. As for the other machines, they are assigned the same set or a subset of the jobs they were assigned in $x^{(2)}$ and thus their completion time is also at most $C \cdot (1 + \delta)$. □

6.2.2 Proving the Cost-Monotonicity condition

In order to prove the Cost-Monotonicity condition, we show that $\textsc{Oracle}_\delta$ uses all and only the fastest machines. This is because every solution using more machines than $\textsc{Oracle}_\delta$ incurs a larger cost:

Lemma 20 *Every solution of cost at most $C \cdot (1 + \delta)$ assigns a positive work to at most $f(C, t, W)$ machines, for every instance (t, W) and for $f(C, t, W)$ being defined as in Phase III of $\textsc{Oracle}_\delta$.*

Proof. Let x be a solution that assigns positive work to $k > f(C, t, W)$ machines. We show that its cost must be more than $C \cdot (1 + \delta)$. Without loss of generality, we can assume that x uses only the k fastest machines (otherwise there is another solution that does it and whose cost is not larger). Moreover, the cost of x is at least the cost of some assignment y for the k smallest jobs such that no two jobs go to the same machine: we modify x by removing jobs until each of the k fastest machines remains with one job and, if the jobs that are left do not contain all of the k smallest jobs, we can swap a (discarded) small job with another (larger) job without increasing the cost. We conclude by showing that the cost of y obtained in this way is at least the cost of $matching^{(k)}$. This is because in any y such that every machine gets at most one job, we can always swap to jobs W_a and $W_b \geq W_a$, that are assigned to machines of speeds $s(a)$ and $s(b) \leq s(a)$, without increasing the makespan. So, the assignment $matching^{(k)}$ can be obtained in this way from y without increasing the cost of y. We conclude that the cost of x is at least the cost of $matching^{(k)}$. By definition of $f(C, t, W)$ and since $k > f(C, t, W)$, the cost of $matching^{(k)}$ is larger than $C \cdot (1 + \delta)$, and thus the cost of x is more than $C \cdot (1 + \delta)$. □

Notice that the effect of the adjustment phase is to have all fast machines with positive work. The above lemma thus implies the following:

Lemma 21 *If $\textsc{Oracle}_\delta(C, t, W) \neq$ fail then the computed solution assigns a positive work to all and only the $f(C, t, W)$ fastest machines, where $f(C, t, W)$*

is defined as in Phase III of ORACLE$_\delta$.

PROOF. Lemma 19 and the Closeness condition imply that the cost of ORACLE$_\delta(C, t, W)$ is at most $C \cdot (1 + \delta)$. Lemma 20 implies that this solution can assign a positive work to at most $f(C, t, W)$ machines. Moreover, as effect of the adjustment phase, all of the $f(C, t, W)$ fastest machines have positive work. □

We are now in a position to prove the following:

Lemma 22 ORACLE *satisfies the Cost-Monotonicity condition.*

PROOF. Let $x = \text{ORACLE}_\delta(C, t, W) \neq \mathsf{fail}$ and $w(x, i) = 0$. We have to show that, for every $C' < C$ such that $x' = \text{ORACLE}_\delta(C', t, W) \neq \mathsf{fail}$, it holds that $w(x', i) = 0$. According to Lemma 21, solution x assigns a positive work to all of the $f(C, t, W)$ fastest machines and, since $w(x, i) = 0$, machine i is not among them (i.e., there are at least $f(C, t, W)$ machines having a better speed or the same speed but a smaller index). Similarly, solution x' assigns a positive work only to the $f(C', t, W)$ fastest machines. By definition and because of $C' < C$, we have that $f(C', t, W) \leq f(C, t, W)$. So, machine i is not among the $f(C', t, W)$ fastest machines and therefore $w(x', i) = 0$. □

6.2.3 Proving the Stability condition

We can prove the Stability condition by imposing a similar property (only) on the subroutine EXACT used to assign large jobs.

Definition 23 (stable algorithm) *An algorithm is* stable *if, for all $t \in D$ and for all i, the following holds. If $w_i^A(t) = 0$ then $A(t) = A(b_i, t_{-i})$, for all $b_i > t_i$ with $b_i \in D_i$.*

Notice that the above condition strengthen that of roughly monotone algorithm (Definition 5). We shall prove that imposing such a stronger condition only on algorithm EXACT guarantees rough monotonicity on the "entire" algorithm SCAN$_\varepsilon$.

Lemma 24 ORACLE *satisfies the Stability condition if algorithm* EXACT *is stable and it assigns positive work to all and only the fastest machines (i.e., if a machine receives one job, then every faster machine and every machine with smaller index and same speed does).*

PROOF. Let (t, W) be such that $w(x, i) = 0$ with $x = \text{ORACLE}_\delta(C, t, W)$. We let $x^{(a)}$ be the solution computed by ORACLE$_\delta$ at the end of the corresponding phase of ORACLE$_\delta$ on input (t, W). Hence, $w(x^{(3)}, i) = 0$. We prove the theorem by considering the phases of the algorithm separately and show

that the solution computed at the end of each phase does not change on input $((b_i, t_{-i}), W)$, for $b_i > t_i$:

(1) *Phase I.* We first show that, since $w(x^{(3)}, i) = 0$, then $w(x^{(1)}, i) = 0$. Let $e(C, t, W)$ be the number of machines that receive a positive work in $x^{(1)} = \textsc{Exact}(C, t, W)$. Recall that the cost of $x^{(1)}$ with respect to the rounded weights is at most C and thus its cost with respect to the original weights is at most $C \cdot (1+\delta)$. Lemma 20 implies that $e(C, t, W) \leq f(C, t, W)$. Since $w(x^{(3)}, i) = 0$, machine i is not among the $f(C, t, W)$ fastest machines (Lemma 21). Since Exact assigns positive work to all and only the fastest machines, and these are $e(C, t, W)$ many, we conclude that $w(x^{(1)}, i) = 0$.

Finally, since Exact is stable and since $w(x^{(1)}, i) = 0$, we have that $\textsc{Exact}(C, (b_i, t_{-i}), W) = \textsc{Exact}(C, t, W) = x^{(1)}$.

(2) *Phase II.* By the previous argument we know that if $w(x^{(1)}, i) = 0$ then the solution given in Phase I on input b' is also $x^{(1)}$. Moreover, by definition of greedy allocation, if a job is allocated to a non empty machine, then there is no faster machine with no work. We conclude that the assignment $x^{(2)}$ produced at the end of this phase assigns positive work to the fastest machines. Since its cost is at most $C \cdot (1 + \delta)$, with the same argument used in the previous item, we can show that $w(x^{(3)}, i) = 0$ implies $w(x^{(2)}, i) = 0$. This means that every time a small job is considered in Phase II it is not assigned to machine i because another machine j has a better completion time or it has the same completion time and $j < i$. Obviously, the same happens if machine i becomes slower (i.e., for $b_i > t_i$). We conclude that the solution returned at the and of Phase II on input $((b_i, t_{-i}), W)$ is $x^{(2)}$.

(3) *Phase III.* This phase then starts with the same assignment $x^{(2)}$ for both (t, W) and $((b_i, t_{-i}), W)$. We claim that, since $w(x^{(3)}, i) = 0$, then $f(C, t, W) = f(C, (b_i, t_{-i}), W)$. Hence, the adjustment is the same for both (t, W) and $((b_i, t_{-i}), W)$ and so is the final solution, that is, $x^{(3)} = \textsc{Oracle}_\delta(C, (b_i, t_{-i}), W)$.

We have thus shown that $\textsc{Oracle}_\delta(C, t, W) = x^{(3)} = \textsc{Oracle}_\delta(C, (b_i, t_{-i}), W)$, thus implying that the Stability condition holds. □

6.3 Focusing on Algorithm Exact *(proof of the main theorem)*

By putting the results of the previous sections together, we obtain that we only have to focus on algorithm Exact used to assign large jobs optimally in Phase I of $\textsc{Oracle}_\delta$. In particular, the existence of the mechanism simply follows from two properties of Exact, as stated by the following theorem:

Lemma 25 *For every $\varepsilon > 0$, algorithm* $\textsc{Scan}_\varepsilon$ *is a roughly monotone polynomial-time* $(1+\varepsilon)$*–approximation algorithm for* $Q||C_{\max}$ *in the case of bounded speeds if the following two conditions holds:*

(1) Algorithm Exact *is stable and it assigns positive work to all and only the fastest machines. I.e., if the d fastest machine receives no work, for some d, then also the $d-1$ fastest machine receives no work;*
(2) Algorithm Exact *runs in polynomial time on the input of Phase I of* $\textsc{Oracle}_\delta$*, where $\delta = \varepsilon/3$.*

Proof. Lemmas 22 and 24 imply that Oracle satisfies the Cost-Monotonicity and the Stability conditions, respectively. From Theorem 18 we have that $\textsc{Scan}_\varepsilon$ is roughly monotone. Lemma 19 and Theorem 17 imply that $\textsc{Scan}_\varepsilon$ is a $(1+\varepsilon)$–approximation algorithm. It remains to show that $\textsc{Scan}_\varepsilon$ runs in polynomial time. Observe that $\textsc{Scan}_\varepsilon$ invokes $\textsc{Oracle}_\delta$ for $O(\log C^*)$ times, with C^* being the optimum of the instance (t, W) in input to $\textsc{Scan}_\varepsilon$. Since C^* is polynomial in n, m and in the largest number in (t, W), it suffices to show that, each time $\textsc{Scan}_\varepsilon$ invokes $\textsc{Oracle}_\delta$, the latter runs in polynomial time. The running time of Phase I is dominated by the running time of algorithm Exact on the input received by $\textsc{Oracle}_\delta$, which is polynomial by hypothesis. The running time of Phase II is clearly polynomial since jobs are assigned greedily. For Phase III, we have already observed that it requires at most m adjustment steps. □

We conclude by proving the main result of this section which is the existence of a $(1+\varepsilon)$–approximation mechanism for any number of machines in the case of bounded speeds (Theorem 15). In Appendix A, we describe a dynamic-programming algorithm Exact and show the following:

Lemma 26 *Algorithm* Exact *satisfies the two conditions of Lemma 25. In particular, its running time is $O(n \cdot m^{2k} \cdot k)$ where k is the number of distinct rounded weights of the large jobs received by* $\textsc{Oracle}_\delta$*, with $\delta = \varepsilon/3$. For the case of bounded speeds and for every fixed $\varepsilon > 0$ it holds that $k = O(1)$.*

Since $Q||C_{\max}$ is a smooth problem (see Definition 16), Theorem 15 follows from Theorem 13 and Lemmas 25-26.

We conclude this section by discussing a few extensions of Theorem 15.

Remark 27 *We stress that Theorem 15 can be applied to the case in which speeds are in a range $[Min, Max]$, with Min being a known value, while Max does not need to be known to the mechanism. In this case, the definition of "small" jobs in* $\textsc{Oracle}_\delta$ *becomes "of weight at most $\delta \cdot C \cdot Min$". The value of Max affects only the running time of* Exact *(and thus of* $\textsc{Scan}_\varepsilon$*) since $k = O(\log_{1+\varepsilon}(Max/(\varepsilon \cdot Min)))$.*

Remark 28 *It is natural to ask which part (algorithm) needs to be changed (improved) in order to extend the result to "non-constant" speeds. The "constant speeds" assumption is used to guarantee that* Exact *has polynomial running time and that the "greedy assignment" of the small jobs (Phase II of* Oracle$_\delta$*) introduces only a "small error" (Lemma 19). Hence, a* $(1+\varepsilon)$*–approximation mechanism for "non-constant" speeds based on the analysis presented here would require a modification of* Exact *and of Phase II of* Oracle$_\delta$*.*

7 The power of verification

Nisan and Ronen [9] proved that, for scheduling unrelated machines, there is a mechanism with verification whose approximation ratio is better than the approximation ratio of any truthful mechanism without verification.

In this section we show that, even for one-parameter agents, there exist optimization problems for which truthful mechanisms with verification can achieve a provably better approximation factor than any truthful mechanism without verification. Archer and Tardos [2] proved the first lower bound on the approximation ratio achievable by a (possibly super polynomial) truthful mechanism without verification for a specific optimization problem involving one-parameter agents. We first show that for the same optimization problem there exist *exact* truthful mechanisms with verification. We then show analogous results for yet another optimization problem considered in [3]. In this case, we show that a *polynomial-time* mechanism with verification can achieve an approximation factor better than the approximation factor of any mechanism without verification, including those *not* running in polynomial time.

7.1 Weighted sum scheduling

Archer and Tardos [2] considered the well-known $\mathrm{Q}||\sum p_j \mathrm{C}_j$ scheduling problem in the case of selfish machines. This optimization problem is a variant of $\mathrm{Q}||\mathrm{C}_{\max}$ in which the goal is to minimize the *weighted sum* of all jobs completion times.

Definition 29 (Problem input and formulation for $\mathrm{Q}||\sum p_j \mathrm{C}_j$.) *The* public part *of the input* π *consists of a vector* $W = (W_1, \ldots, W_m)$ *of* jobs weights*, and a vector* $p = (p_1, \ldots, p_m)$ *of* jobs priorities*. The private part of the input, the set of feasible solutions, and the work assigned to each agent are defined as in the* $\mathrm{Q}||\mathrm{C}_{\max}$ *problem. For every feasible solution* x*, we denote the completion time of a job* j *as* $C_j(x, t, W)$*. The* optimization function *is*

the weighted sum *of all jobs completion times, that is,* $\text{Measure}(x,t,W,p) = \sum_j p_j \cdot C_j(x,t,W)$. *We assume that each machine executes jobs in nondecreasing order of* $\frac{p_j}{W_j}$ *(this is the well-known Smith's Ratio rule [12]). The Smith's Ratio rule guarantees that, for every assignment of jobs to machines, no other internal scheduling policy results in a better weighted sum of jobs completion times [12]. Thus, a feasible solution* x *is completely specified by the assignment of jobs to machines.*

This scheduling problem shows the reason why mechanisms with verification can achieve an approximation factor that is provably better than the approximation factor of any mechanism without verification. Indeed, the latter mechanisms require *monotone* algorithms [8,2], while verification allows us to relax this condition and use *roughly monotone* algorithms (see Corollaries 8 and 14). Archer and Tardos [2] showed that no monotone algorithm (and thus no truthful mechanism without verification) can achieve an approximation factor better than $2/\sqrt{3}$. In contrast, we can show that *every* exact algorithm for this problem is roughly monotone (and thus a $(1+\varepsilon)$–approximation can be achieved by mechanisms with verification).

Theorem 30 *Every exact algorithm for* $\text{Q}||\sum p_j \text{C}_j$ *is roughly monotone.*

Proof. Let A^* be any exact algorithm for $\text{Q}||\sum p_j \text{C}_j$ and denote by $A(b)$ the solution output by A^* on input (b,W,p), for arbitrary W and p. We proceed by contradiction and assume that there exist b and $b' = (b_i, b_{-i})$, with $b'_i > b_i$, such that $w_i^A(b) = 0$ and $w_i^A(b') > 0$.

Since $b_i > b'_i$ and $w_i^A(b') > 0$, we have that

$$\text{Measure}(A(b'), b', W, p) > \text{Measure}(A(b'), b, W, p).$$

Since $A(b)$ is optimal for (b,W,p), we have that

$$\text{Measure}(A(b'), b, W, p) \geq \text{Measure}(A(b), b, W, p).$$

Since $w_i^A(b) = 0$, we have that

$$\text{Measure}(A(b), b, W, p) = \text{Measure}(A(b), b', W, p).$$

Putting things together we obtain

$$\text{Measure}(A(b'), b', W, p) > \text{Measure}(A(b), b', W, p),$$

thus contradicting the optimality of $A(b') = A^*(b', W, p)$ for the instance (b', W, p). □

The above result indicates that roughly monotone algorithms define a "natural" class since it contains all exact algorithms for $\text{Q}||\sum p_j \text{C}_j$. More impor-

tantly, from this result we can easily derive optimal truthful mechanism with verification for $Q||\sum p_j C_j$, in contrast to the lower bound for mechanisms (without verification) in [2].

Corollary 31 *For every $\varepsilon > 0$, there exists a $(1+\varepsilon)$–approximation truthful mechanism with verification for $Q||\sum p_j C_j$. Moreover, for finite speeds (i.e., the set of possible machine speeds is finite) there exists an exact truthful mechanism with verification. By contrast, there is no c–approximation truthful mechanism without verification for the $Q||\sum p_j C_j$ problem, for every $c < 2/\sqrt{3}$ [2]. This negative result holds also for the case of two machines with finite speeds and mechanisms that do not (necessarily) run in polynomial time.*

Proof. For finite speeds, the domain is finite and thus Corollary 8 implies the existence of an exact truthful mechanism with verification. The existence of a $(1+\varepsilon)$–approximation mechanism for the case of arbitrary speeds is obtained from Corollary 14. Indeed, notice that exact algorithms are bounded since the no optimal solution assigns a job to a machine if its speed is sufficiently small. Moreover, the problem is smooth because a "small" change in the processing time of one machine has only a "small" impact of the jobs completion times. This proves the upper bounds and completes the proof. □

We remark that a "small error" ε is introduced only to guarantee truthfulness for the infinite domain resulting from arbitrary speeds. Although our mechanism with verification does not (necessarily) run in polynomial time, its approximation factor is better than the approximation factor guaranteed by any truthful mechanism without verification, including those running in exponential time. Actually, this lower bound applies to finite speeds as well, in which case we obtain *exact* truthful mechanisms with verification.

In the next section, we will consider another optimization problem for which we can obtain computationally-efficient mechanisms. In particular, we will show that *polynomial-time* mechanisms with verification can achieve an approximation factor better than any mechanism without verification, even allowing *exponential* running time.

7.2 Scheduling selfish jobs

Consider the case in which jobs represent *traffic demands* and the machines are *carriers* of different speeds. A natural goal is to minimize the maximum "congestion", that is, the makespan (no carrier is "overloaded" and no traffic demand is delayed "too much"). The resulting problem is a sort of "dual" of the $Q||C_{\max}$ problem with selfish machines in the sense that we consider *selfish jobs* instead. So, the speeds are known to the mechanism, while the size

(weights) of the jobs are the private part of the input. Intuitively, each agent prefers the solutions that result is a smaller completion time for her own job, regardless of the completion time of the other jobs and/or the global cost, i.e., the makespan.

Here we consider a natural scenario in which the mechanism can "observe" the amount of traffic sent by each agent. In particular, an agent whose job has weight $W_i = t_i$ can always report a weight $b_i \geq t_i$ and then actually submit a job of weight b_i (i.e., send the original traffic padded with some fake data). Conversely, if such an agent declares a weight $b_i < t_i$, then the agent is caught lying since she has to send an amount of traffic at least t_i (i.e., the agent needs to transmit all of her traffic and cannot split it among several machines). This is a variant of the problem in [3] where the authors consider mechanisms *without* verification. That is, the case in which the mechanism cannot observe the amount of data the agents transmit (equivalently, the payments received depend only on the agents bids). For this scenario several lower bounds are known [3]. We will show that mechanisms with verification can break also these lower bounds. Let us first define formally the problem:

Definition 32 (Problem input and formulation for selfish jobs.) *The* public part *of the input π consists of a vector $s = (s_1, \ldots, s_m)$ of* machine speeds*. The private part of the input is a vector $W = (W_1, \ldots, W_n)$ of n job weights, one for each agent. Hence, the type t_i of agent i is $t_i = W_i$. The set of feasible solutions consists of all job assignments to machines. In particular, every solution x assigns a subset $x(j)$ jobs to machine j. Thus, machine j's completion time is $\sum_{i \in x(j)} t_i/s_j$. Machines execute jobs in a round-robin fashion and all jobs assigned to the same machine are completed in (roughly) the same amount of time. Each agent valuates a solution x as $-C_i(x, t, s)$, where $C_i(x, t, s) \stackrel{\text{def}}{=} \sum_{k \in x(j)} t_k/s_j$ is the completion time of job i. The optimization function is the makespan.*

Notice that, unlike the case of one-parameter agents, the valuation of an agent depends also on the types t_{-i} of the other agents. This crucial aspect of the problem leads us to consider mechanisms that are truthful with respect to Bayesian-Nash equilibrium (as done in [3]). These mechanisms guarantee that truthtelling is the best strategy for every agent i, provided that all the other agents are truthtelling:

Definition 33 *A mechanism (with verification) for selfish jobs is* truthful with respect to Nash equilibria *(in short,* NE–truthful*) if truthtelling is a Nash equilibrium. That is, for all $t \in D$, for all i, and for all $b_i \in D_i$, it holds that*

$$P_i^{ver}(t|t_i) - C_i(x, t, s) \geq P_i^{ver}(b_i, t_{-i}|t_i) - C_i(x', t, s),$$

where (A, P) is the mechanism under consideration, $x = A(t)$ and $x' = A(b_i, t_{-i})$.

Notice that every truthful mechanism is also a NE-truthful mechanism, while the converse does not hold in general. The reason is that in a truthful mechanism truthtelling is a *dominant* strategy, that is, the utility of agent i is maximized also when the other agents are not truthtelling.

In the sequel, we will "reduce" the construction of NE-truthful mechanisms for selfish jobs to the construction of truthful mechanism for a "related" one-parameter agents problem. Towards this end, we introduce a natural variant of the problem in which each agent i is only allowed to send up to b_i units of traffic and, if an agent transmits only $t_i < b_i$ units, the resources (time slots) allocated to her are *not* reallocated to the other agents. The important thing here is that, from the point of view of agent i, the machine on which her job has been allocated "looks like" being executing her job of size t_i together with other jobs of size b_j. This is formally captured by this definition:

Definition 34 *The case of selfish jobs with* restricted access *is defined as the problem in Definition 32 with the difference that the completion time of job i is equal to $\bar{C}_i(x,t,b,s) \stackrel{\text{def}}{=} t_i/s_j + \sum_{k\in x(i),k\neq i} b_k/s_j$, where j is the machine to which x assigns job i and b is the bid vector. The valuation of agent i is equal to $-\bar{C}_i(x,t,b,s)$ and thus it does not depend on the types t_{-i} of the other agents. To distinguish the case of jobs with restricted access from the one in Definition 32 we denote the latter also as the case of selfish jobs with* unrestricted access.

In the case of unrestricted access, truthful mechanisms are "difficult" to obtain because the valuation of agent i depends also on the types t_{-i} of the other agents. The variant of selfish jobs with restricted access allows us to circumvent this problem and makes the valuation of i depending only on her type t_i. The following result shows the connection between the restricted access and the unrestricted access cases:

Theorem 35 *Every truthful mechanism with verification for selfish jobs with restricted access is a NE-truthful mechanism with verification for selfish jobs with unrestricted access.*

Proof. Let (A,P) be a truthful mechanism with verification for selfish jobs with restricted access. For every $t,b \in D$ and for every i, the mechanism guarantees that

$$P_i^{ver}(t_i,b_{-i}|t_i) - \bar{C}_i(x,t,b,s) \geq P_i^{ver}(b_i,b_{-i}|t_i) - \bar{C}_i(x',t,b,s)$$

where $x = A(t_i,b_{-i})$ and $x' = A(b_i,b_{-i})$. In particular, for every $t \in D$, for every i, and for every $b_i \in D_i$, we have $\bar{C}_i(y,t,(b_i,t_{-i}),s) = t_i/s_j + \sum_{k\in y(j),k\neq i} t_k/s_j = C_i(y,t,s)$, where y is an arbitrary solution and j is the machine to which y assigns job i. The above inequality then implies the one in Definition 33. That is, the same mechanism (A,P) is NE-truthful for selfish

jobs with unrestricted access. □

In [3] it is proved that, even for two machines, no NE–truthful mechanism can guarantee an approximation better than $\frac{1+\sqrt{17}}{4}$. This, combined with Theorem 35, yields the following result:

Corollary 36 *No truthful mechanism without verification for scheduling selfish jobs with restricted access can achieve an approximation factor better than* $\frac{1+\sqrt{17}}{4}$. *This negative result holds also for the case of two machines and for mechanisms that do not (necessarily) run in polynomial time.*

In contrast, we will show that a $(1+\varepsilon)$–approximation factor can be achieved by polynomial-time mechanisms with verification. We first show the following general result:

Theorem 37 *Every (polynomial-time) c–approximation algorithm for* $\mathrm{Q}||\mathrm{C}_{\max}$ *can be turned into a (polynomial-time) truthful $c(1+\varepsilon)$–approximation mechanism with verification for scheduling selfish job with restricted access, for every* $\varepsilon > 0$.

Proof. We show that every (polynomial-time) c–approximation algorithm A is "essentially" a polynomially-bounded algorithm for an "equivalent" smooth problem involving one-parameter agents. With the same technique used to prove Theorem 13, we obtain a polynomial-time $c(1+\varepsilon)$–approximation mechanism with verification $(\hat{A}, \hat{P})$ for this "equivalent" problem. We modify the payments $\hat{P}$ and obtain another mechanism with verification $(\hat{A}, \hat{Q})$ which is truthful for scheduling selfish job with restricted access. We stress that we cannot use directly the result of Theorem 13 since the "work" associated to an agent will be *always strictly positive*. Strictly speaking, no algorithm can be bounded, but having a non-zero work ensures a "stronger" verification: An underbidding agent is always caught lying and does not receive her payment.

Since the problem is smooth (a "small" change in the size t_i of a job has only a "small" impact on the makespan), we can round the bids in input to some power of $\gamma = \gamma(\varepsilon)$ and obtain in this way a $c(1+\varepsilon)$–approximation algorithm $\hat{A}$. For every solution x, we define $s(x,i)$ as the speed of the machine to which x assigns job i. Let us consider $w(x,i) \stackrel{\text{def}}{=} 1/s(x,i)$. Notice that this defines a problem with one-parameter agents in which the "work" $w(x,i)$ is in the interval $[1/s_{\max}, 1/s_{\min}]$ and thus it is always strictly positive. Without loss of generality, we can assume that our algorithm A assigns a job i to the fastest machine as soon as the input b satisfies $b_i \geq \sum_{k\neq i} b_k$ (otherwise we can swap the assignment of the fastest with that of the machine containing job i without increasing the cost). Therefore, when considering the rounded input $\hat{b}$, we have that $w_i^{\hat{A}}(b'_i, b_{-i}) = w_{\min} \stackrel{\text{def}}{=} 1/s_{\max}$ for all $b'_i \geq B_i(\hat{b}_{-i}) \stackrel{\text{def}}{=} \gamma \sum_{k\neq i} \hat{b}_k$. We observe that this condition suffices to apply the construction of the payments

in the proof of Theorem 13. In particular, the same arguments show that the resulting payments $\hat{P}$ guarantee truthfulness for overbidding agents if we restrict the bids to the interval $(0, B_i(\hat{b}_{-i})]$. More precisely, for every b_{-i}, for every $t_i \leq B_i(\hat{b}_i)$, and for every $b_i \leq B_i(\hat{b}_{-i})$ with $b_i > t_i$, the following inequality can be guaranteed:

$$\hat{P}_i(t_i, b_{-i}) - w_i^{\hat{A}}(t_i, b_{-i}) \cdot t_i \geq \hat{P}_i(b). \tag{26}$$

The only difference with the proof of Theorem 13 is that this inequality cannot be extended to $b_i > B_i(\hat{b}_{-i})$ since now we have $w_i^{\hat{A}}(b_i, b_{-i}) = w_{\min} > 0$. So, using the same arguments of that proof, we impose (16) only for rounded bids in $(0, B_i(\hat{b}_{-i})]$ and obtain that the above inequality holds for b_i and t_i in $(0, B_i(\hat{b}_{-i})]$. We extend these payments to arbitrary bids in the natural way, that is, by setting $\hat{P}_i(b'_i, b_{-i}) = \hat{P}_i(B_i(\hat{b}_{-i}), b_{-i})$ for all $b'_i > B_i(\hat{b}_{-i})$.

Now we go back to the problem of scheduling selfish jobs and define the following payments:

$$\hat{Q}_i(b) \stackrel{\text{def}}{=} \hat{P}_i(b) + \sum_{k \in x(i), k \neq i} w_i^{\hat{A}}(b) \cdot b_k.$$

We show that the resulting mechanism $(\hat{A}, \hat{Q})$ is a truthful mechanism with verification for scheduling selfish jobs with restricted access. The key observation is that, if agent i is not caught lying for bids b, then a solution $x = \hat{A}(b)$ is computed and her utility is equal to

$$\begin{aligned}
\hat{Q}(b) - \bar{C}_i(\hat{A}(b), t, b, s) &= \hat{Q}(b) - t_i/s(\hat{A}(b), i) - \sum_{k \in x(i), k \neq i} b_k/s(\hat{A}(b), i) && (27)\\
&= \hat{Q}(b) - w_i^{\hat{A}}(b) \cdot t_i - \sum_{k \in x(i), k \neq i} w_i^{\hat{A}}(b) \cdot b_k && (28)\\
&= \hat{P}(b) - w_i^{\hat{A}}(b) \cdot t_i. && (29)
\end{aligned}$$

In particular, the utility of agent i when reporting any $b_i \geq t_i$ with $b_i \geq B_i(\hat{b}_{-i})$ is equal to

$$\hat{P}_i(B_i(\hat{b}_{-i}), b_{-i}) - w_{\min} \cdot t_i.$$

Moreover, if agent i reports any $b_i < t_i$, then she is caught lying and receives no payment. In this case the utility is equal to

$$-t_i/s(\hat{A}(b), i) \leq -t_i/s_{\min} = -w_{\min} \cdot t_i.$$

Since the payments are nonnegative, this shows that underbidding is never the best strategy for agent i and that we can restrict the analysis to the case $t_i < b_i \leq B_i(\hat{b}_{-i})$. Here we can simply apply (26) and obtain

$$\hat{P}_i(t_i, b_{-i}) - w_i^{\hat{A}}(t_i, b_{-i}) \cdot t_i \geq \hat{P}_i(b) > \hat{P}_i(b) - w_i^{\hat{A}}(b).$$

Notice that the left and the right hand side is the utility of agent i when reporting t_i and b_i, respectively. We conclude that $(\hat{A}, \hat{Q})$ is a truthful mechanism with verification for scheduling selfish jobs with restricted access. If the algorithm A is runs in polynomial time, then the payments $\hat{P}$ can be computed in polynomial time. □

The existence of a PTAS for $Q||C_{max}$ [4] and the theorem above imply the following result which provides a sharp contrast between mechanisms with verification and mechanisms without verification (Corollary 36).

Corollary 38 *There exists a polynomial-time $(1+\varepsilon)$-approximation truthful mechanism with verification for scheduling selfish jobs with restricted access, for every $\varepsilon > 0$.*

The same results can be transposed to the case of selfish jobs with unrestricted access, studied in [3], by considering NE-truthful mechanisms.

Corollary 39 *There exists a polynomial-time $(1+\varepsilon)$-approximation NE–truthful mechanism with verification for scheduling selfish jobs with unrestricted access, for every $\varepsilon > 0$. By contrast, no NE–truthful mechanism without verification can achieve an approximation factor better than $\frac{1+\sqrt{17}}{4}$ [3]. This negative result holds also for the case of two machines and for mechanisms that do not (necessarily) run in polynomial time.*

8 Conclusions and open questions

In this work we initiate the study of mechanisms with *verification* for *one-parameter* agents. We give an algorithmic *characterization* of such mechanisms and show that they are *provably better* than mechanisms without verification for a number of optimization problems motivated by the Internet and recently studied in the algorithmic mechanism design literature. The characterization can be regarded as an alternative approach to existing techniques to design *truthful* mechanisms. The construction of a (computationally efficient) mechanism reduces to the construction of an (computationally efficient) roughly monotone algorithm which satisfies a few more "natural" conditions (see Theorem 13).

The results presented in this paper relates to a number of works in (algorithmic) game theory. Mechanisms with "partial verification" have been studied by Singh and Wittman [11]. Their model implicitly assumes that verification is done "a priori" and the type of the agent automatically restricts the possible "lies" that she could tell the mechanism. More precisely, an agent of type t_i is restricted to report a type in a subset $R(t_i)$ of the domain, for some function

$R(\cdot)$. The natural application of this model to scheduling problems would be to say that an agent could only report a "worse" type, that is, a higher cost per unit of work. This is precisely the case of overbidding agents, which we consider in Section 3, that is $R(t_i) = \{b_i \in D_i |\ b_i \geq t_i\}$. The characterization that we obtain in Section 4 shows that the model in [11] does not capture the fact that no verification is possible if an agent gets no work.

There is also an interesting difference between our model and the formal setting given in [9] for unrelated machines. In order to understand this difference, consider the scenario in which the "principal" (mechanism) assign certain amount of work each "worker" (agent). In the Nisan and Ronen setting, workers will "suffer" the amount of time they have to sit on their desk pretending that they are doing the assigned work (i.e., until they communicate to the principal that work has been done along with the results). In our setting, instead, a worker "suffers" as the amount of time that it really takes to her to complete the work. Pretending a longer time will not cost more as the "idle" time could be used for other activities (e.g., if the worker is doing her job at home). We feel this as an important difference to consider in future research since computing facilities are better modeled by the latter scenario (CPU idle time is of no cost and/or can be reused for other purposes, as well as other unused resources). Also, truthful mechanisms for our setting are also truthful in the other one (i.e., when the cost for an agent is exactly the completion time of the given work).

Our results suggest a number of questions which are still open. First of all, it would be interesting to extend the main positive result to more general domains. It is also open the existence of a *polynomial-time* $(1+\varepsilon)$-approximation mechanism *without verification* for the makespan in the case of arbitrary number of machines. The question is whether verification helps for related machines. Can we extend our result on this problem to the case of speeds not bounded by a constant? More generally, what is the best approximation ratio of a *computationally efficient* mechanism for a given problem? Does verification help in finding computationally efficient mechanisms with better approximation? Noticeably, existing lower bounds do not use computational assumptions (i.e., $P \neq NP$). For the problem of minimizing the jobs weighted completion time, one would like to be able to achieve provably better approximations in *polynomial time*, as we do already for the case of selfish jobs. The study of approximation classes for optimization problems involving selfish agents is certainly an interesting topic. This relates to two important aspects like the agents domains and to the possibility of using verification (for one-parameter agents, whether the algorithm has to be only roughly monotone or monotone).

References

[1] N. Andelman, Y. Azar, and M. Sorani. Truthful approximation mechanisms for scheduling selfish related machines. In *Proc. of the Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 69–82, 2005.

[2] A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2001.

[3] V. Auletta, R. De Prisco, P. Penna, and P. Persiano. How to route and tax selfish unsplittable traffic. In *Proc. of the Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 196–205, 2004. To appear in *ACM Transactions on Algorithms*.

[4] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[5] E. Koutsoupias and C.H. Papadimitriou. Worst-case equilibria. In *Proc. of Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 of *LNCS*, pages 404–413, 1999.

[6] A. Kovács. Fast monotone 3-approximation algorithm for scheduling related machines. In *Proc. of Annual European Symposium on Algorithms (ESA)*, pages 616–627, 2005.

[7] A. Mu'alem and M. Schapira. Setting lower bounds on truthfulness. In *Proc. of Annual ACM Symposium on Discrete Algorithms (SODA)*, pages 1143–1152, 2007.

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

[9] N. Nisan and A. Ronen. Algorithmic Mechanism Design. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 129–140, 1999.

[10] Christos H. Papadimitriou. Algorithms, games, and the internet. In *Proc. of Annual ACM Symposium on Theory of Computing (STOC)*, pages 749–753, 2001.

[11] N. Singh and D. Wittman. Implementation with partial verification. *Review of Economic Design*, 6(1):63–84, 2001.

[12] W.E. Smith. Various optimizer for single-stage production. *Naval Research and Logistic Quaterly*, 3:59–66, 1956.

[13] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

A Dynamic programming EXACT algorithm

In this appendix we describe a simple dynamic programming algorithm EXACT and prove Lemma 26. This algorithm receives in input a set $W = \{W_1, \ldots, W_m\}$ of m jobs, n machines of processing times $t = (t_1, \ldots, t_n)$ and a bound C. We let k be the number of distinct weights for the jobs in W and consider the set $\{V_1, \ldots, V_k\}$ of such weights. Algorithm EXACT checks in time $O(n \cdot m^{2k} \cdot k)$ whether there exists an assignment of makespan at most C. If such an assignment exists, then it returns the "lexicographically minimal" one to ensure that all and only the fastest machines receive positive work (see Item 1 in Lemma 25). Otherwise, it simply returns another assignment whose cost is larger than C. We shell see that k is bounded by some constant when the set of jobs is the one provided by ORACLE_δ, thus proving Lemma 26.

Definition 40 *We denote every subset S of W as a k-tuple $\mathbf{s} = (s^{(1)}, \ldots, s^{(k)})$ where $s^{(h)}$ is the number of jobs in S whose weight is V_h. For every subset R of S, we denote the set $S \setminus R$ by $\mathbf{s} - \mathbf{r} \stackrel{\text{def}}{=} (s^{(1)} - r^{(1)}, \ldots, s^{(k)} - r^{(k)})$. For $0 \leq d \leq n-1$ and for any $\mathbf{s}$ as above, we define $\text{BINS}(C, \mathbf{s}, t, d)$ to take value 0 if there exists a solution of cost at most C for the set of jobs $\mathbf{s}$ that assigns no work to the d slowest machines (according to the processing times $t = (t_1, \ldots, t_n)$). If such a scheduling does not exist, then $\text{BINS}(C, \mathbf{s}, t, d)$ equals to 1. Finally, we let $\text{STORE}(C, \mathbf{s}, t, d)$ denote the set of tuples $\mathbf{r}$ such that $R \subseteq S$ and the $d+1$ slowest machine can complete all jobs in R in time at most C.*

Clearly, there exists a scheduling for the jobs W of cost at most C if and only if

$$\text{BINS}(C, \mathbf{w}, t, 0) = 0.$$

To compute this value observe that $\text{BINS}(C, \mathbf{s}, t, n-1) = 0$ if and only if it is possible to execute all jobs in $\mathbf{s}$ on the fastest machine in time not greater than C (we assign no work to the $n-1$ slowest machines). Instead, for $d < n$ we have

$$\text{BINS}(C, \mathbf{s}, t, d) = \min_{\mathbf{r} \in \text{STORE}(C, \mathbf{s}, t, d)} \text{BINS}(C, \mathbf{s} - \mathbf{r}, t, d+1). \tag{A.1}$$

This identity can be easily verified by observing the following. The set $\mathbf{r}$ is the (possibly empty) subset of jobs that we assign to the $d+1$ slowest machine, and thus $\mathbf{s} - \mathbf{r}$ is the set of jobs that have to be assigned to the remaining machines, that is, none of these jobs is assigned to the $d+1$ slowest machine. In both cases, we have to guarantee that the makespan is at most C and thus the identity follows from Definition 40.

Notice that (A.1) also suggests how to compute the corresponding assignment. Indeed, it suffices to keep track of the vector $\mathbf{r}^*$ such that $\text{BINS}(C, \mathbf{s}, t, d) =$

$\textsc{Bins}(C, \mathbf{s} - \mathbf{r}^*, t, d+1)$ and $\mathbf{r}^* \in \textsc{Store}(C, \mathbf{s}, t, d)$. Such an $\mathbf{r}^*$ is associated to $\textsc{Bins}(C, \mathbf{s}, t, d)$ meaning that the $d+1$ slowest machine is assigned the subset $\mathbf{r}^*$ and the other jobs are assigned as in the solution associated to $\textsc{Bins}(C, \mathbf{s} - \mathbf{r}^*, t, d+1)$. In order to guarantee that Exact assigns positive work to the fastest machines, we break ties in a fixed manner. In particular, we assign the vector $\mathbf{0} \stackrel{\text{def}}{=} (0, \ldots, 0)$, denoting the empty set of jobs, to $\textsc{Bins}(C, \mathbf{s}, t, d)$ whenever $\textsc{Bins}(C, \mathbf{s}, t, d) = \textsc{Bins}(C, \mathbf{s}, t, d+1) = 0$ (notice that $\mathbf{0} \in \textsc{Store}(C, \mathbf{s}, t, d)$ and $\mathbf{s} - \mathbf{0} = \mathbf{s}$).

The overall computation proceeds from $d = n - 1$ down to 1 and it ensures the following fundamental condition:

Lemma 41 *Algorithm* Exact *is stable and it assigns positive work to the fastest machines.*

Proof. Let $d^* = d^*(C, \mathbf{w}, t)$ be the maximum d such that $\textsc{Bins}(C, \mathbf{w}, t, d) = 0$. For every $d < d^*$, we have $\textsc{Bins}(C, \mathbf{w}, t, d) = 0$ and, by our fixed tie breaking rule, the solution associated to $\textsc{Bins}(C, \mathbf{w}, t, d)$ is the same as the solution associated to $\textsc{Bins}(C, \mathbf{w}, t, d^*)$. We next show that this solution assigns positive work to all but the d^* slowest machines. Indeed, $\textsc{Bins}(C, \mathbf{w}, d^* + 1) = 1$, thus implying that any solution which assigns no work to at least $d^* + 1$ machines has cost more than C. Since $\textsc{Bins}(C, \mathbf{w}, t, d^*) = 0$, its associated solution has cost at most C and thus must assign positive work to at least $n - d^*$ machines. By definition, this solution assigns no work to the d^* slowest machines and thus each of the $n - d^*$ fastest machines receives a positive work. □

Lemma 42 *The running time of* Exact *is* $O(n \cdot m^{2k} \cdot k)$.

Proof. From (A.1) we derive a simple method to compute $\textsc{Bins}(C, \mathbf{s}, t, d)$ given the values $\textsc{Bins}(C, \mathbf{s}', t, d+1)$ for all $\mathbf{s}'$. Indeed, it suffices to try all possible $\mathbf{r}$, which are at most $O(m^k)$, check $\mathbf{r} \in \textsc{Store}(C, \mathbf{s}, t, d)$ (this task takes $O(k)$ time), and decide in this way if $\textsc{Bins}(C, \mathbf{s}, t, d) = 0$. So, for every d, we compute the $O(m^k)$ possible values $\textsc{Bins}(C, \mathbf{s}, t, d)$, one for each $\mathbf{s}$. Each of them can be computed in $O(m^k \cdot k)$, given the pre-computed values for $d+1$. The overall running time is thus $O(n \cdot m^{2k} \cdot k)$. Recall that our algorithm will also keep track of the computations in order to reconstruct the assignment (i.e., it records all $\mathbf{r}$ such that (A.1) holds). In doing this, it usees the afore mentioned fixed tie breaking rule where $\mathbf{r} = \mathbf{0}$ is used whenever possible. Obviously the running time remains $O(n \cdot m^{2k} \cdot k)$. □

The previous lemma proves the first part of Lemma 26. In order to prove the second part, we show that $k = O(1)$ for the case of bounded speeds and for every fixed $\varepsilon > 0$. By our rounding and by the definition of large jobs, the rounded jobs can take $k \in O\left(\log_{1+\varepsilon} \frac{W_{\max}}{\varepsilon C}\right)$ different values. Since $\textsc{Oracle}_\delta$ invokes Exact only for C, t, and W satisfying $C \geq W_{\max} \cdot t_{\min}$, where $t_{\min}$

denotes the smallest element in t, we have $k \in O\left(\log_{1+\varepsilon} \frac{1}{\varepsilon t_{\min}}\right)$. The hypothesis on the machine speeds says that there exists a constant $\overline{s}$ such that $s_i \in [1, \overline{s}]$. That is, $t_i = 1/s_i \geq t_{\min} \geq 1/\overline{s}$, thus implying $k \in O\left(\log_{1+\varepsilon} \frac{\overline{s}}{\varepsilon}\right)$. We conclude that $k = O(1)$ for bounded speeds and for fixed $\varepsilon > 0$, thus yielding Lemma 26.