To appear in Proc. of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 06) July 9 - 16, 2006 S. Servolo, Venice - Italy

New Constructions of Mechanisms with Verification

Vincenzo Auletta[1] Roberto De Prisco[1] Paolo Penna[1] Giuseppe Persiano[1]
Carmine Ventre[1]

April 20, 2006

[1] Dipartimento di Informatica ed Applicazioni
Università di Salerno
Via S. Allende 2, 84081 Baronissi (SA), Italy
Email: {auletta,robdep,penna,giuper,ventre}@dia.unisa.it
URL: http://www.dia.unisa.it/~{auletta,robdep,penna,giuper,ventre}

Abstract

A social choice function A is implementable with verification if there exists a payment scheme P such that (A, P) is a truthful mechanism for verifiable agents [Nisan and Ronen STOC 99]. In this paper we address the following questions. Given an objective function μ, does there exist a social choice function that is *implementable with verification* and that minimizes (or maximizes) μ? From a more algorithmic point of view, we ask whether there exists an *efficient* social choice function A that is implementable with verification and α-approximates μ.

We give a simple sufficient condition for a social choice function to be implementable with verification for *comparable* types. Comparable types are a generalization of the well studied one-parameter agents. Based on this characterization, we show that a large class of objective functions μ admit social choice functions that are implementable with verification and minimize (or maximize) μ.

We then focus on the well-studied case of one-parameter agents. We give a general technique for constructing *efficiently computable* social choice functions that minimize or approximately minimize objective functions that are *non-increasing* and *neutral* (these are functions that do not depend on the valuations of agents that have no work assigned to them). Our results are based on a reduction of problems with one-parameter agents to combinatorial auctions with *known single-minded* bidders.

As a corollary we obtain efficient mechanisms with verification for some hard scheduling problems on related machines. Finally, we show that existing online algorithms for scheduling jobs to minimize the L_p norm can be used to obtain *online* mechanisms with verification.

Keywords: algorithmic game theory, truthful mechanisms, implementation theory, approximation algorithms.

Work supported by EU under Integrated Project AEOLUS.

Contents

1 Introduction

Computations over the Internet often involve self-interested parties (*selfish agents*) which may manipulate the system by misreporting a fundamental piece of information they hold (their own *type* or *valuation*). The system runs some algorithm which, because of the misreported information, is no longer guaranteed to return a "globally optimal" solution (optimality is naturally expressed as a function of agents' types) [19, 18]. Since agents can manipulate the algorithm by misreporting their types, one has to carefully design payment functions which make disadvantageous for an agent to do so. A *mechanism* $M = (A, P)$ consists of a *social choice function* A which, on input the reported types, chooses an outcome, and a payment function P which, on input the reported types, associate a payment to every agent. Payments should guarantee that it is in the agent's interest to report his type correctly. Social choice functions A for which there exists a payment P that guarantees that the *utility* that an agent derives from the chosen outcome and from the payment he receives is maximum when this agent reports his type correctly are called *implementable* (see Sect. 2 for a formal definition of these concepts). In this case the mechanism $M = (A, P)$ is called *truthful*. The main difficulty in designing truthful mechanisms stems from the fact that the utility itself depends on the type of the agent: for instance, payments designed to "compensate" certain costs of the agents should make impossible for an agent to speculate. It is well-known that certain social choice functions cannot be implemented. This poses severe limitations on the class of optimization problems involving selfish agents that one can optimally solve (see e.g. [18, 1]).

This paper aims at deriving a general technique for building optimal (and close to optimal) social choice functions for a given optimization problem involving selfish agents. In particular, we focus on so called mechanisms with *verification* as introduced by Nisan and Ronen [18]. These mechanisms award payments *after* the selected outcome has been "implemented" and this implementation allows some limited "verification" on the agents' types. By contrast, more "classical" mechanisms without verification award the payment associated to an agent unconditionally (i.e., without performing any kind of verification and solely based on the agents reported types).

There are two main reasons for being interested in mechanisms with verification. First of all, there are specific optimization problems for which verification allows to overcome certain impossibility results for mechanisms without verification [18, 3]. Moreover, mechanisms with verification are very natural and many real-life applications require and/or already implement this kind of approach: reputation is used by the e-Bay system to ensure that sellers actually ship the goods as they have been offered in the auction; service providers offer connectivity to the Internet with the guarantee of a minimal rate.

2 Implementation with verification

In this section we review the concept of a social choice function *implementable with verification*. The following notations will be useful. For a vector $\mathbf{x} = (x_1, \ldots, x_m)$, we let $\mathbf{x}_{-i}$ denote the vector $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m)$ and $(y, \mathbf{x}_{-i})$ the vector $(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_m)$. For sets $D_1, \cdots, D_m$, we let D denote the Cartesian product $D_1 \times \cdots \times D_m$ and, for $1 \leq i \leq m$, we let D_{-i} denote the Cartesian product $D_1 \times \cdots \times D_{i-1} \times D_{i+1} \times \cdots \times D_m$.

We have a finite set $\mathcal{O}$ of possible outcomes and m selfish rational agents. Agent i has a *valuation* (or *type*) v_i taken from a finite set D_i called the *domain* of agent i. A valuation v_i is a function $v_i : \mathcal{O} \to \Re$; $v_i(X)$ represents how much agent i likes outcome $X \in \mathcal{O}$ (higher valuations correspond to preferred outcomes). The valuation v_i is known to agent i only. A *social choice function* $A : D \to \mathcal{O}$ maps the agents' valuations into a particular outcome $A(v_1, \ldots, v_m)$.

A *mechanism* $M = (A, P)$ is a social choice function A coupled with a *payment scheme* $P = (P_1, \ldots, P_m)$, where each P_i is a function $P_i : D \to \Re$. The mechanism elicits from each agent its valuation and we denote by $b_i \in D_i$ the *reported* valuation of agent i. On input the vector $\mathbf{b} = (b_1, \ldots, b_m)$ of *reported* valuations, the mechanism selects outcome X as $X = A(\mathbf{b})$ and assigns agent i payment $P_i(\mathbf{b})$. We assume that agents have quasi-linear utilities; more specifically, the *utility* $u_i^M(\mathbf{b}|v_i)$ of agent i when $\mathbf{b}$ is the vector of reported valuations and v_i is the type of agent i is $u_i^M(\mathbf{b}|v_i) = P_i(\mathbf{b}) + v_i(A(\mathbf{b}))$. Agents are selfish and rational in the sense that they will report b_i which maximizes their utility.

We stress that both the outcome and the payments depend on the *reported* valuations $\mathbf{b} = (b_1, \ldots, b_m)$. In particular, for a fixed $\mathbf{b}_{-i}$, the outcome $A(\mathbf{b}_{-i}, b_i)$ is a function $A_{\mathbf{b}_{-i}}(b_i)$ of the reported valuation b_i

of agent i.

The classical notion of a mechanism assumes that there is no way of verifying whether an agent reported his type faithfully (that is, whether $b_i = v_i$). Therefore, a selfish rational agent can declare any type that will maximize his utility. In some cases, though, it is reasonable to assume that the mechanism has some limited way of verifying the reported types of the agents. In this paper, we consider *mechanisms with verification* which can detect whether $b_i \neq v_i$ if and only if $v_i(A_{\mathbf{b}_{-i}}(b_i)) < b_i(A_{\mathbf{b}_{-i}}(b_i))$; in this case, agent i will *not* receive any payment.

MOTIVATION. A scenario that is often considered when dealing with selfish rational agents consists of a social choice function that has to share some work-load among the agents. In this scenario, an outcome X specifies for each agent the task that the agent has to complete. It is thus natural to assume that the valuation $v_i(X)$ of agent i reflects how much time it takes agent i to complete the task assigned to him. For example, one could have $v_i(X) = -T_i(X)$ where $T_i(X)$ is the time needed by agent i to complete the task assigned to him by X; thus higher valuations correspond to outcomes X that assign to agent i tasks that can be completed faster. In this scenario, it is natural to assume that an agent can report to be slower than he actually is and delay the completion of the task assigned to him without being caught by the mechanism (this corresponds to the case in which agent i declares b_i such that $b_i(A_{\mathbf{b}_{-i}}(b_i)) \leq v_i(A_{\mathbf{b}_{-i}}(b_i))$. On the other hand, if agent i declares to be faster that he actually is (that is, he declares b_i such that $b_i(A_{\mathbf{b}_{-i}}(b_i)) > v_i(A_{\mathbf{b}_{-i}}(b_i))$) then agent i will complete his task at time $-v_i(A_{\mathbf{b}_{-i}}(b_i))$ instead of time $-b_i(A_{\mathbf{b}_{-i}}(b_i))$ as expected by the mechanism, given his declared valuation b_i. The mechanism will thus punish agent i by not giving him any payment.

For example, the well-studied class of one-parameter agents [17, 1] corresponds to the special case in which the task assigned to an agent is described by a weight and the time needed to complete a task is proportional to its weight. In this case, the type of the agent is determined by the time it takes the agent to complete a task of unitary weight. Let us now proceed more formally.

Definition 1 ([18]) *A social choice function A is* implementable with verification *if there exists $P = (P_1, \ldots, P_m)$ such that for all i, all $v_i \in D_i$, all $\mathbf{b}_{-i} \in D_{-i}$, utility $u_i^{(A,P)}(\mathbf{b}|v_i)$ of agent i is maximized by setting $b_i = v_i$.*

In this case, $M = (A, P)$ is called a *truthful mechanism with verification*.

It is easy to see that, if A is implementable with verification then there exists $P = (P_1, \ldots, P_m)$ such that, for all $v_i, b_i \in D_i$ and $\mathbf{b}_{-i} \in D_{-i}$, the following inequalities hold:

$$v_i(A_{\mathbf{b}_{-i}}(v_i)) + P_i(v_i, \mathbf{b}_{-i}) \geq v_i(A_{\mathbf{b}_{-i}}(b_i)) \quad \text{if } v_i(A_{\mathbf{b}_{-i}}(b_i)) < b_i(A_{\mathbf{b}_{-i}}(b_i)), \tag{1}$$

$$v_i(A_{\mathbf{b}_{-i}}(v_i)) + P_i(v_i, \mathbf{b}_{-i}) \geq v_i(A_{\mathbf{b}_{-i}}(b_i)) + P_i(b_i, \mathbf{b}_{-i}) \quad \text{if } v_i(A_{\mathbf{b}_{-i}}(b_i)) \geq b_i(A_{\mathbf{b}_{-i}}(b_i)). \tag{2}$$

The notion of implementable with verification we use in this paper corresponds to the notion of truthfulness with respect to dominant strategies when verification is allowed. We are interested in social choice functions A which are implementable with verification and that optimize some *objective function* $\mu(\cdot)$ which depends on the agent valuations $\mathbf{v} = (v_1, \ldots, v_m)$. For maximization (respectively, minimization) functions, we let $\text{OPT}_\mu(\mathbf{v})$ be $\max_{X \in \mathcal{O}} \mu(X, \mathbf{v})$ (respectively, $\min_{X \in \mathcal{O}} \mu(X, \mathbf{v})$). An outcome $X \in \mathcal{O}$ is an α-approximation of μ for $\mathbf{v} \in D$ if

$$\max\left\{\frac{\mu(X, \mathbf{v})}{\text{OPT}_\mu(\mathbf{v})}, \frac{\text{OPT}_\mu(\mathbf{v})}{\mu(X, \mathbf{v})}\right\} \leq \alpha.$$

A social choice function A is *α-approximate* for μ if, for every $\mathbf{v} \in D$, $A(\mathbf{v})$ is an α-approximation of μ for $\mathbf{v}$. In particular, we say that social choice function A maximizes function μ if, for all $\mathbf{v}$,

$$A(\mathbf{v}) = \arg\max_{X \in \mathcal{O}} \mu(X, \mathbf{v}).$$

We stress that, in this paper we consider social choice functions that are implementable with verification and either optimize or α-approximate a function μ. The approximation only refers to how good the selected outcome is and not to the utilities of the agents (which are always maximized by reporting the true valuation).

3 Our results

In this work we study social choice functions that are *implementable with verification*. Implementation with verification has been first studied by Nisan and Ronen [18].

We start by studying a generalization of one-parameter agents which we call *comparable* types. We give a simple sufficient condition for social choice function to be implementable with verification for comparable types and, based on this characterization, we show that a large class of objective functions μ admit social choice functions that are implementable with verification and minimize (or maximize) μ. In particular, we consider maximization (respectively, minimization) functions of the form $\mu(v_1(X), \ldots, v_m(X))$ which are monotone non-decreasing (respectively, non-increasing) in each agent valuation $v_i(X)$. Observe that VCG mechanisms [25, 8, 12] can only deal with particular functions of this form called *affine maximizers* (basically, the case $\mu(v_1(X), \ldots, v_m(X)) = \sum_i \beta_i v_i(X)$, with constants β_i defined by the mechanisms). The $Q||C_{\max}$ scheduling problem is an example of an optimization problem involving a monotone non-decreasing function (thus our result applies to $Q||C_{\max}$) that is not an affine maximizer. We also show that, for agents with comparable types, verification is indeed helpful, since there exists a social choice function that maximizes a monotone non-decreasing function (that by our results is implementable with verification) which cannot be implemented if verification is not allowed (see Section 4.2).

We remark that agents with comparable types are more general than one-parameter agents. Consider for example the case in which agents own router and the valuation of each agent is the latency of his router. One-parameter agents can only model situations in which the latency depends linearly with the load. Instead with comparable types we can model any class of latency functions; the only restriction is that for any two latency functions, it is always the case that one is "better" than the other for all possible loads. The example in Section 4.2 shows that there exists such a class of latencies for which optimization is not implementable if verification is not allowed.

We also *characterize* social choice functions implementable with verification for agents with *strongly comparable types*, a reach subclass of comparable types which has the well-studied one-parameter agents as a special instance. Our characterization (Theorem 16) extends the characterization result by Auletta *et al* [3] for one-parameter agents.

In Section 5, the focus is on *efficiently* computable social choice functions and one-parameter agents. We consider social choice functions that optimize monotone functions μ that depend on *the outcome* and on the valuations of the agents; that is, $\mu = \mu(X, v_1, \ldots, v_m)$ instead of $\mu = \mu(v_1(X), \ldots, v_m(X))$ as in Section 4. We give a general transformation for turning any polynomial-time α-approximate algorithm A for the optimization problem of function μ into an $\alpha(1+\varepsilon)$-approximate social choice function $A^\star$ that is implementable with verification. If the number of agents is constant, $A^\star$ can be computed in polynomial-time.

An immediate application of our results yields polynomial-time optimal approximation mechanisms with verification for several NP-hard scheduling problems with one-parameter agents (see Section 6). In particular, for any constant number of machines, our polynomial-time $(1+\varepsilon)$-approximate mechanisms with verification for $Q_m||\sum \mathrm{w_j C_j}$ and $Q_m|\mathrm{r_j}|\sum \mathrm{w_j C_j}$ break the $2/\sqrt{3} \approx 1.154$ lower bound by Archer and Tardos [1] which holds for mechanisms without verification, even if we allow exponential running time and consider two machines/agents only. Our constant-approximation for $Q_m|prec_j, \mathrm{r_j}|\sum \mathrm{w_j C_j}$ matches the approximation guarantee of the best known algorithm and better algorithms, if any, yield better approximation mechanisms (with roughly the same algorithm's approximation factor). Finally, a constant approximation is obtained for scheduling problems in which we want to minimize the L_p norm via a simple *online* greedy algorithms.

Related work. For agents with quasi-linear utility functions (i.e., payment received plus agent's monetary valuation), the celebrated VCG mechanism [25, 12, 8] show that affine maximizers (see above) can be implemented and Roberts [22] showed that VCG mechanisms are essentially the only truthful mechanisms if no hypothesis is made on the domains of the agents. Mechanisms for *one-parameter* agents have been characterized in [17, 1]. Lavi, Mu'alem and Nisan [15] showed that a *weak monotonicity* condition (W-MON) characterizes *order-based* domains with range constraints and this result was extended, in a sequence of papers [13, 24], to *convex domains*. These results concern mechanisms which do *not* use verification and cannot be applied to our case (indeed, one wishes to use mechanisms with verification

to solve problems which the other mechanisms cannot solve [18, 3]). We show that the "counterpart" of W-MON for mechanisms with verification (which we term W-MON-VER) is not always sufficient, unlike the cases considered in [15, 13, 24]. This gives evidence that the results about W-MON cannot be imported in mechanisms with verification.

The study of social choice functions implementable with verification starts with the work of Nisan and Ronen [18], who gave a truthful $(1+\varepsilon)$-approximate mechanism for minimizing scheduling on a constant number of unrelated machines. Similar results have been obtained by Auletta *et al.* [3] for scheduling on any number of related machines (see also [2] for the online case). These results are based on a characterization of mechanisms with verification [3]. Also the works of [14, 20] give mechanisms for agents which are verifiable.

The work of [5] presents a general technique for constructing polynomial-time approximation mechanisms for *utilitarian* problems. This technique is similar to the one we use to obtain truthful mechanisms with verification for one-parameter agents: Indeed both approaches are derived from [16], where agents' types are even simpler than the one-parameter case. Polynomial-time mechanisms which approximate the social welfare for certain auctions are given in [9]. Mechanisms in [5, 9, 10] do not use verification but the problems are utilitarian. We are interested in truthful mechanisms with verification but we do not restrict ourselves to utilitarian problems (see for example the applications to non-utilitarian scheduling problems).

4 Agents with Comparable Types

In this section we consider *comparable types*. The main result of this section (see Theorem 9) shows that, for any monotone non-decreasing function μ, there exists a social choice function A that maximizes μ and that is implementable with verification. We also prove that verification is essential to achieve this result as there are social choice functions A that maximize monotone non-decreasing functions and that cannot be implemented if verification is not allowed. Finally, in Theorem 16, we give a necessary and sufficient condition for a social choice function to be implementable with verification with respect to a subclass of comparable types which includes one-parameter agents.

Definition 2 *Let a and b be valuations. We say that a is* smaller or equal to *b, in symbols $a \leq b$, if, for all $X \in \mathcal{O}$, $a(X) \leq b(X)$.*

Definition 3 *Domain D is* comparable *if for any $a, b \in D$ either $a \leq b$ or $b \leq a$.*

In this section we assume that for all i, the domain D_i of agent i is comparable. We also assume domains to have finite cardinality (even though this assumptions can be relaxed in some cases, e.g., for one-parameter agents).

Discussion. In the scenario discussed in the previous section, the valuation of an agent is a measure of his cost to perform the task assigned to him. If an agent has "better capabilities" then his costs are lower and the valuation is higher (e.g., costs can be measured as the processing time which this agent has to dedicate to the computations that X assigns to her). Thus if $a \leq b$, a type b agent has better capabilities than a type a agent. Among others, we can model the following situations:

- An agent whose type is a has a subset of the resources of an agent whose type is b;
- each agent has resources of r different kinds (e.g., memory, CPU speed, and bandwidth), and each resource of an agent with valuation b is at least as good as the corresponding resource of an agent with valuation a (e.g., b's memory is higher than a's memory, while CPU speed and bandwidth are the same); for instance we can model scheduling problems in which a job requires time $t^a \geq t^b$ on machine a while t^b is processing time of machine b.

In a mechanism with verification, an agent can misreport his type by declaring a worse one (i.e., $b_i \leq v_i$) and escape any "verification" by just making his capabilities look worse (e.g., simulating lower CPU speed, releasing jobs later, delaying traffic, etc.). The converse (i.e., $b_i \geq v_i$) is also possible whenever the selected outcome X makes b_i "indistinguishable" from v_i, that is, $b_i(X) = v_i(X)$. (Observe that the agent could prefer to declare b_i because of a higher payment.)

For fixed i and $\mathbf{b}_{-i}$, inequalities (1-2) give a system of linear inequalities with unknowns $P^x := P_i(x, \mathbf{b}_{-i})$, for $x \in D_i$. By expanding inequalities (1-2) for $a, b \in D_i$ with $a \leq b$ we obtain

$$\begin{array}{lcll} P^a & \geq & a(A_{\mathbf{b}_{-i}}(b)) - a(A_{\mathbf{b}_{-i}}(a)) & \text{if } a(A_{\mathbf{b}_{-i}}(b)) < b(A_{\mathbf{b}_{-i}}(b)), \quad (3) \\ P^a - P^b & \geq & a(A_{\mathbf{b}_{-i}}(b)) - a(A_{\mathbf{b}_{-i}}(a)) & \text{if } a(A_{\mathbf{b}_{-i}}(b)) \geq b(A_{\mathbf{b}_{-i}}(b)), \quad (4) \\ P^b & \geq & b(A_{\mathbf{b}_{-i}}(a)) - b(A_{\mathbf{b}_{-i}}(b)) & \text{if } b(A_{\mathbf{b}_{-i}}(a)) < a(A_{\mathbf{b}_{-i}}(a)), \quad (5) \\ P^b - P^a & \geq & b(A_{\mathbf{b}_{-i}}(a)) - b(A_{\mathbf{b}_{-i}}(b)) & \text{if } b(A_{\mathbf{b}_{-i}}(a)) \geq a(A_{\mathbf{b}_{-i}}(a)). \quad (6) \end{array}$$

Observe that, since $a \leq b$, we have: (i) the condition of Inequality 4 is equivalent to $a(A_{\mathbf{b}_{-i}}(b)) = b(A_{\mathbf{b}_{-i}}(b))$; (ii) the condition of Inequality 6 is always satisfied; (iii) and the condition of Inequality 5 is never satisfied. Moreover, by assuming that payments are nonnegative (which we can always enforce), if payments satisfy Inequality 4 then Inequality 3 is satisfied as well. Therefore we can conclude that social choice function A is implementable with verification if and only if, for each i and $\mathbf{b}_{-i} \in D_{-i}$ and $a, b \in D_i$ with $a \leq b$, Inequalities (7,8) are satisfied.

$$\begin{array}{lcll} P^a - P^b & \geq & a(A_{\mathbf{b}_{-i}}(b)) - a(A_{\mathbf{b}_{-i}}(a)) & \text{if } a(A_{\mathbf{b}_{-i}}(b)) = b(A_{\mathbf{b}_{-i}}(b)), \quad (7) \\ P^b - P^a & \geq & b(A_{\mathbf{b}_{-i}}(a)) - b(A_{\mathbf{b}_{-i}}(b)). & \quad (8) \end{array}$$

For fixed i and $\mathbf{b}_{-i}$ the two inequalities above give rise to a system of inequalities as a and b with $a \leq b$ range over D_i. This system of inequalities is compactly encoded by the following graph.

Definition 4 (verification-graph) *Let A be a social choice function. For every i and $\mathbf{b}_{-i} \in D_{-i}$, the* verification-graph $\mathcal{V}(\mathbf{b}_{-i})$ *has a node for each type in D_i. The set of edges of $\mathcal{V}(\mathbf{b}_{-i})$ is defined as follows. For every $a \leq b$, add a directed edge (b, a) of weight $\delta_{b,a} := b(A_{\mathbf{b}_{-i}}(b)) - b(A_{\mathbf{b}_{-i}}(a))$. This edge encodes Inequality (8). If $a(A_{\mathbf{b}_{-i}}(b)) = b(A_{\mathbf{b}_{-i}}(b))$, then also add directed edge (a, b) of weight $\delta_{a,b} := a(A_{\mathbf{b}_{-i}}(a)) - a(A_{\mathbf{b}_{-i}}(b))$. This edge encodes Inequality (7).*

The definition of the verification-graph is a modification of the graph introduced in [13] to study the case in which verification is not allowed. The graph of [13] has an edge of weight $\delta_{a,b}$ between *all* types a and b. Instead in the verification-graph the edge between a and b is missing, if $a \leq b$ and $a(A_{\mathbf{b}_{-i}}(b)) < b(A_{\mathbf{b}_{-i}}(b))$.

Theorem 5 *A social choice function A is implementable with verification if and only if, for all i and $\mathbf{b}_{-i} \in D_{-i}$, the graph $\mathcal{V}(\mathbf{b}_{-i})$ does not have negative weight cycles.*

The theorem follows from the observation that the system of linear inequalities involving the payment functions is the linear programming dual of the shortest path problem on the verification-graph. Therefore, a simple application of Farkas lemma shows that the system of linear inequalities has solution if and only if the verification-graph has no negative weight cycle. The same argument has been used for the case in which verification is not allowed albeit of a different graph (see [23] and [13]).

4.1 Optimal social choice functions

In this section we show that there exists an interesting class of social choice functions whose graphs have no cycle with negative weights that be used to design optimal truthful mechanisms with verification.

Definition 6 (stable social choice function) *A social choice function A is* stable *if, for all i, for all $\mathbf{b}_{-i} \in D_{-i}$, and for all $a, b \in D_i$, with $a \leq b$, if $a(A_{\mathbf{b}_{-i}}(b)) = b(A_{\mathbf{b}_{-i}}(b))$, then we have that $A_{\mathbf{b}_{-i}}(a) = A_{\mathbf{b}_{-i}}(b)$.*

The following result is based on the fact that stable social choice functions guarantee that, if $\mathcal{V}(\mathbf{b}_{-i})$ contains a cycle, then all edges in that cycle have zero weight.

Theorem 7 *Every stable social choice function A is implementable with verification.*

Proof. We show that if A is stable all edges of $\mathcal{V}(\mathbf{b}_{-i})$ that are part of a cycle have zero weight.

We start by showing that, if $\mathcal{V}(\mathbf{b}_{-i})$ contains the edge (a, c) with $a \leq c$, then it also contains the edge (b, c) for all $a \leq b \leq c$. Indeed edge (a, c) exists only if $a(A_{\mathbf{b}_{-i}}(c)) = c(A_{\mathbf{b}_{-i}}(c))$. However, as $a \leq b \leq c$,

we have that $a(A_{\mathbf{b}_{-i}}(c)) \leq b(A_{\mathbf{b}_{-i}}(c)) \leq c(A_{\mathbf{b}_{-i}}(c))$ and thus $b(A_{\mathbf{b}_{-i}}(c)) = c(A_{\mathbf{b}_{-i}}(c))$ which implies the existence of edge (b, c).

Let $C = c_1 \to c_2 \to \cdots \to c_k \to c_1$ be a cycle in $\mathcal{V}(\mathbf{b}_{-i})$. If no such cycle exists, then the theorem holds. Since C is a cycle, for every vertex $b \in C$ there must exist a, c such that $a \leq b \leq c$ and (a, c) is an edge in $\mathcal{V}(\mathbf{b}_{-i})$. Notice that a and c, but not both, may coincide with b. By the argument above, we have that edge (b, c) belongs to $\mathcal{V}(\mathbf{b}_{-i})$ which implies that $a(A_{\mathbf{b}_{-i}}(c)) = c(A_{\mathbf{b}_{-i}}(c))$ and $b(A_{\mathbf{b}_{-i}}(c)) = c(A_{\mathbf{b}_{-i}}(c))$. By the stability of A we conclude that $A_{\mathbf{b}_{-i}}(a) = A_{\mathbf{b}_{-i}}(b) = A_{\mathbf{b}_{-i}}(c)$. Hence, for any two vertices c_i and c_j in C, it holds that $A_{\mathbf{b}_{-i}}(c_i) = A_{\mathbf{b}_{-i}}(c_j)$, thus implying that every edge in the cycle C has zero weight. □

We use the above result to show that it is possible to implement social choice functions which select the best outcome out of a fixed subset of possible outcomes:

Theorem 8 *For any $X_1, \ldots, X_\ell \in \mathcal{O}$, let $A = MAX_\mu(X_1, \ldots, X_\ell)$ be the social choice function that, on input $(b_1, \ldots, b_m) \in D$, returns the solution X_j of minimum index that maximizes the value*

$$\mu(b_1(X_j), \ldots, b_m(X_j)).$$

If $\mu(\cdot)$ is monotone non-decreasing in each of its arguments then A is stable.

Proof. Fix an agent i and the reported types $\mathbf{b}_{-i} \in D_{-i}$ of all the other agents. Let $a, b \in D_i$ with $a \leq b$, and denote $X_{i_a} := A_{\mathbf{b}_{-i}}(a)$ and $X_{i_b} := A_{\mathbf{b}_{-i}}(b)$. To prove that A is stable we have to show that, if $a(A_{\mathbf{b}_{-i}}(b)) = b(A_{\mathbf{b}_{-i}}(b))$, then $X_{i_a} = X_{i_b}$. Observe that

$$\begin{aligned}
&\mu(b_1(X_{i_b}), \ldots, b_{i-1}(X_{i_b}), b(X_{i_b}), \ldots, b_m(X_{i_b})) && = && \text{(by } a(X_{i_b}) = b(X_{i_b})) && (9)\\
&\mu(b_1(X_{i_b}), \ldots, b_{i-1}(X_{i_b}), a(X_{i_b}), \ldots, b_m(X_{i_b})) && \leq && \text{(definition of } A \text{ and } X_{i_a}) && (10)\\
&\mu(b_1(X_{i_a}), \ldots, b_{i-1}(X_{i_a}), a(X_{i_a}), \ldots, b_m(X_{i_a})) && \leq && (a \leq b \text{ and } \mu \text{ non decr.}) && (11)\\
&\mu(b_1(X_{i_a}), \ldots, b_{i-1}(X_{i_a}), b(X_{i_a}), \ldots, b_m(X_{i_a})) && \leq && \text{(definition of } A \text{ and } X_{i_b}) && (12)\\
&\mu(b_1(X_{i_b}), \ldots, b_{i-1}(X_{i_b}), b(X_{i_b}), \ldots, b_m(X_{i_b})). && && && (13)
\end{aligned}$$

This implies that all inequalities above hold with "=". Since A chooses the optimal solution of minimal index, equality between (9) and (12) yields $i_b \leq i_a$. Similarly, the equality between (11) and (10) yields $i_a \leq i_b$, thus implying $X_{i_a} = X_{i_b}$. □

Combining Theorem 8 and Theorem 7 we obtain the main result of this section.

Theorem 9 *Let $\mu(\cdot)$ be any monotone non-decreasing function in its arguments $b_1(X), \ldots, b_m(X)$, with $X \in \mathcal{O}$ and $b_i \in D_i$. Then, there exists a social choice function OPT_μ which maximizes $\mu(\cdot)$ and is implementable with verification.*

4.2 The power of verification

We next exhibit a social choice function which satisfies the hypothesis of Theorem 8 (and thus is implementable with verification) but is not implementable if verification is not allowed. This shows that, for comparable types, verification does help.

We have one agent that owns one router which can be of one of two types. A type a_1 router has latency $L_1(\cdot)$ equal to 1 (respectively 3) for transmitting one packet (respectively two packets); a type a_2 router has latency $L_2(\cdot)$ equal to 0 (respectively 1) for transmitting one packet (respectively two packets). Assume $\mathcal{O} = \{X, Y\}$ with Y (respectively X) being an outcome that assigns 1 (respectively 2) packet(s) to the router and $X \prec Y$ in a given lexicographic order. Valuations a_1 and a_2 are defined to be the opposite of the latency and can be found in the following table:

	X	Y
$a_1(\cdot) = -L_1(\cdot)$	-3	-1
$a_2(\cdot) = -L_2(\cdot)$	-1	0

Observe that $a_1(S) \leq a_2(S)$ for $S \in \mathcal{O}$.

Let A be the social choice function that picks the assignment that guarantees latency less than 1 and, in case of a tie, picks the lexicographically smaller assignment (that is X). Therefore, $A(a_1) = X$ and $A(a_2) = Y$. Social choice function A maximizes function μ defined as

$$\mu(v) = \begin{cases} -1, & \text{if } v \leq -1; \\ 0, & \text{otherwise;} \end{cases}$$

which is non-decreasing in its input and thus, by Theorem 8, A is implementable with verification. Function μ encodes a QoS objective function: we are only interested in whether the assignment can be completed within 1 unit of time. Next we show that A is not implementable if verification is not allowed. For A to be implementable we have to find payments P^{a_1} and P^{a_2} such that for $v \in D_1$ the utility of the agent is maximized when reporting $b = v$. Therefore for $v = a_1$ we have that

$$P^{a_1} - 3 \geq P^{a_2} - 1$$

and for $v = a_2$ we have

$$P^{a_2} \geq P^{a_1} - 1.$$

The two inequalities imply $P^{a_1} - 3 \geq P^{a_1} - 2$. Contradiction.

If the set $\mathcal{O}$ of outcomes is very large, then social choice function A could not be efficiently computable. Our next result can be used to derive efficiently-computable social choice functions which approximate the objective function by restricting to a suitable subset of the possible outcomes.

Definition 10 (approximation preserving) *A set $\mathcal{O}' \subseteq \mathcal{O}$ is* α-approximation preserving for μ *if, for every $\mathbf{b} \in D$, the set $\mathcal{O}'$ contains a solution X' which is an α-approximation of μ for $\mathbf{b}$.*

Theorem 8 implies the following.

Corollary 11 *Let $\mu(\cdot)$ be any optimization function monotone non-decreasing in its arguments $b_1(X), \ldots, b_m(X)$, with $X \in \mathcal{O}$ and $b_i \in D_i$. For any α-approximation preserving set $\mathcal{O}'$ the social choice function $APX_\mu := MAX_{X \in \mathcal{O}'}\{X\}$ is an α-approximation for μ and is implementable with verification. Moreover, social choice function $APX_\mu(\mathbf{b})$ can be computed in time proportional to the time needed for computing values $\mu(X, \mathbf{b})$, for $X \in \mathcal{O}'$.*

4.3 W-MON-VER social choice functions

The following definition is adapted to the verification setting from a similar definition of [15].

Definition 12 (W-MON-VER) *A social choice function A is* W-MON-VER *for domains $D_1, \ldots, D_m$, if, for all i, for all $\mathbf{b}_{-i} \in D_{-i}$, the graph $\mathcal{V}(\mathbf{b}_{-i})$ does not contain 2-cycles of negative weight.*

Obviously, condition W-MON-VER is necessary for A to be implementable with verification and is equivalent to the following condition: for all i, for all $\mathbf{b}_{-i} \in D_{-i}$, for all $a, b \in D_i$ with $a \leq b$, if $a(A_{\mathbf{b}_{-i}}(a)) + b(A_{\mathbf{b}_{-i}}(b)) < a(A_{\mathbf{b}_{-i}}(b)) + b(A_{\mathbf{b}_{-i}}(a))$, then $a(A_{\mathbf{b}_{-i}}(b)) < b(A_{\mathbf{b}_{-i}}(b))$. An analogous condition for implementation without verification (called W-MON in [15]) has been proved necessary and sufficient for convex domains [24] (extending the work of [15]). Agents with convex domains include one-parameter agents.

We show that W-MON-VER is not sufficient for A to be implementable with verification for comparable types. We do so by giving an example of a W-MON-VER social choice function for which the associated graph $\mathcal{V}(\mathbf{b}_{-i})$ contains a negative cycle.

Theorem 13 *For comparable types, there exists a W-MON-VER social choice function A that is not implementable with verification. This also holds for domains containing three comparable types over a set of three possible outcomes.*

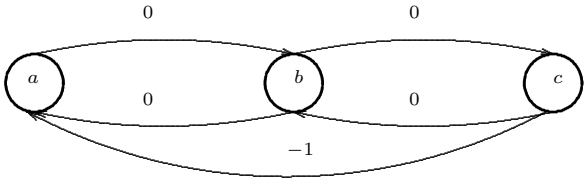

Figure 1: The graph of a social choice function that is W-MON-VER and that is not implementable with verification.

PROOF. Fix agent i and declarations $\mathbf{b}_{-i}$. Let $D_i := \{a, b, c\}$, $\mathcal{O} := \{X_1, X_2, X_3\}$ and let the valuations of agent i be defined as in the following table

	X_1	X_2	X_3
a	1	1	0
b	1	1	1
c	2	1	1

Social choice function is such that $A_{\mathbf{b}_{-i}}(a) = X_1$, $A_{\mathbf{b}_{-i}}(b) = X_2$ and $A_{\mathbf{b}_{-i}}(c) = X_3$. Clearly, agent i has comparable types. We now show that the graph $\mathcal{V}(\mathbf{b}_{-i})$ has a negative cycle (see Figure 1). First of all, observe that the edge (a, c) does not appear in $\mathcal{V}(\mathbf{b}_{-i})$; indeed, we have $a(X_3) = 0 < 1 = c(X_3)$. On the other hand, $\mathcal{V}(\mathbf{b}_{-i})$ contains edge (a, b) (since $a(X_2) = b(X_2)$) and edge (b, c) (since $b(X_3) = c(X_3)$). Moreover, the weight of the edges are computed as follows

$$\begin{array}{rcccc} \delta_{a,b} & = & a(X_1) - a(X_2) & = & 0 \\ \delta_{b,a} & = & b(X_2) - b(X_1) & = & 0 \\ \delta_{b,c} & = & b(X_2) - b(X_3) & = & 0 \\ \delta_{c,b} & = & c(X_3) - c(X_2) & = & 0 \\ \delta_{c,a} & = & c(X_3) - c(X_1) & = & -1 \end{array}$$

We conclude the proof by observing that the graph $\mathcal{V}(\mathbf{b}_{-i})$ has no negative weight 2-cycle (and thus the social choice function is W-MON-VER) while the cycle $a \to b \to c \to a$ has negative weight (and thus A is not implementable with verification). □

We can also provide an example of a W-MON-VER social choice function that is not implementable with verification over types that only take value 0 or 1.

Theorem 14 *For comparable types, there exists a W-MON-VER social choice function A that is not implementable with verification. This also holds for domains containing two comparable types which take values 0 or 1 over a set of three possible outcomes.*

PROOF. Fix agent i and declarations $\mathbf{b}_{-i}$. Let $D_i := \{a, b, c\}$ and let the valuation of agent i be the following

	X_1	X_2	X_3
a	0	1	0
b	0	1	1
c	1	1	1

where $X_1 = A_{\mathbf{b}_{-i}}(a)$, $X_2 = A_{\mathbf{b}_{-i}}(b)$ and $X_3 = A_{\mathbf{b}_{-i}}(c)$. Clearly, agent i has comparable types. We now show that the graph $\mathcal{V}(\mathbf{b}_{-i})$ has a negative cycle (but no two-cycle has negative length). First of all observe that the edge (a, c) does not appear in $\mathcal{V}(\mathbf{b}_{-i})$; indeed, we have $a(X_3) = 0 < 1 = c(X_3)$. On the other hand, $\mathcal{V}(\mathbf{b}_{-i})$ contains edge (a, b) (since $a(X_2) = b(X_2)$) and edge (b, c) (since $b(X_3) = c(X_3)$). Moreover, the weight of the edges are computed as follows

$$\begin{array}{rcccc} \delta_{a,b} & = & a(X_1) - a(X_2) & = & -1 \\ \delta_{b,a} & = & b(X_2) - b(X_1) & = & 1 \\ \delta_{b,c} & = & b(X_2) - b(X_3) & = & 0 \\ \delta_{c,b} & = & c(X_3) - c(X_2) & = & 0 \\ \delta_{c,a} & = & c(X_3) - c(X_1) & = & 0 \end{array}$$

We complete the proof by observing that the graph $\mathcal{V}(\mathbf{b}_{-i})$ has no negative weight 2-cycle (and thus the social choice function is W-MON-VER) while the cycle $a \to b \to c \to a$ has negative weight (and thus A is not implementable with verification). □

4.4 Strongly comparable types

Next we prove that for strongly comparable types (a restriction of comparable types that includes one-parameter types) W-MON-VER is a necessary and sufficient condition for a social choice-function A to be implementable with verification.

Definition 15 (strongly comparable types) *A domain with comparable types D_i is* with strongly comparable types *if there exists $\overline{v}_i \in \Re$ such that, for all $X \in \mathcal{O}$: (i) $a(X) \leq \overline{v}_i$, for all $a \in D_i$, and (ii) for all $a, b \in D_i$, $a(X) = b(X)$ implies $a(X) = \overline{v}_i$.*

We have the following theorem.

Theorem 16 *For domains with strongly comparable types, social choice function A is implementable with verification if and only if A is W-MON-VER.*

Obviously, if the social choice function A is not W-MON-VER then by Theorem 5, A is not implementable with verification. Thus the following lemma is sufficient for proving Theorem 16.

Lemma 17 *If types are strongly comparable and A is W-MON-VER, then, for all i and all $\mathbf{b}_{-i}$, the graph $\mathcal{V}(\mathbf{b}_{-i})$ does not contain a negative weight cycle.*

PROOF. Fix agent i and declarations $\mathbf{b}_{-i}$. We start by proving that all edges $\mathcal{V}(\mathbf{b}_{-i})$ that belong to a 2-cycle have weight 0. Consider the 2-cycle formed by edges (a, b) (of weight $a(A_{\mathbf{b}_{-i}}(a)) - a(A_{\mathbf{b}_{-i}}(b))$) and edge (b, a) (of weight $b(A_{\mathbf{b}_{-i}}(b)) - b(A_{\mathbf{b}_{-i}}(a))$) with $a \leq b$. Since edge (a, b) appears in $\mathcal{V}(\mathbf{b}_{-i})$ we have that $a(A_{\mathbf{b}_{-i}}(b)) = b(A_{\mathbf{b}_{-i}}(b))$ and thus, since types are strongly comparable, $a(A_{\mathbf{b}_{-i}}(b)) = b(A_{\mathbf{b}_{-i}}(b)) = \bar{v}_i$. Therefore the cycle has weight $a(A_{\mathbf{b}_{-i}}(a)) - b(A_{\mathbf{b}_{-i}}(a))$. Since $a \leq b$ it holds that $a(A_{\mathbf{b}_{-i}}(a)) - b(A_{\mathbf{b}_{-i}}(a)) \leq 0$ and by the fact that A is W-MON-VER we have that $a(A_{\mathbf{b}_{-i}}(a)) - b(A_{\mathbf{b}_{-i}}(a)) \geq 0$. The two inequalities imply $a(A_{\mathbf{b}_{-i}}(a)) = b(A_{\mathbf{b}_{-i}}(a))$ and thus $a(A_{\mathbf{b}_{-i}}(a)) = b(A_{\mathbf{b}_{-i}}(a)) = \bar{v}_i$. This implies that the two edges have weight 0.

Next we prove that all edges (b, a) with $a \leq b$ that belong to a cycle and such that the edge (a, b) does not appear in $\mathcal{V}(\mathbf{b}_{-i})$ have non-negative weight. This completes the proof of the lemma.

Let (b, a) be such an edge. We show that $b(A_{\mathbf{b}_{-i}}(b)) = \bar{v}_i$. Since (b, a) is part of a cycle there must exist an edge (c, d) with $c \leq b \leq d$. Notice that c and d, but not both, may coincide with b. If $b = d$ then edge (c, d) implies $c(A_{\mathbf{b}_{-i}}(d)) = d(A_{\mathbf{b}_{-i}}(d)) = b(A_{\mathbf{b}_{-i}}(b)) = \bar{v}_i$. Otherwise, the existence of (c, d) implies that $c(A_{\mathbf{b}_{-i}}(d)) = d(A_{\mathbf{b}_{-i}}(d))$. Since $c \leq b \leq d$, we have that $b(A_{\mathbf{b}_{-i}}(d)) = d(A_{\mathbf{b}_{-i}}(d))$ which implies the existence of edge (b, d). Since $b \leq d$ the edges (d, b) and (b, d) constitute a 2-cycle and, by the argument above, both have weight 0. That is, $d(A_{\mathbf{b}_{-i}}(d)) = d(A_{\mathbf{b}_{-i}}(b)) = \bar{v}_i$ and $b(A_{\mathbf{b}_{-i}}(b)) = b(A_{\mathbf{b}_{-i}}(d)) = \bar{v}_i$.

We conclude the proof by observing that the edge (b, a) has weight $b(A_{\mathbf{b}_{-i}}(b)) - b(A_{\mathbf{b}_{-i}}(a)) = \bar{v}_i - b(A_{\mathbf{b}_{-i}}(a)) \geq 0$ since types are strongly comparable (in particular by definition of $\bar{v}_i$). □

The above theorem is *tight* as the counterexample in Figure 1 shows that there exist types that are not strongly comparable for which a W-MON-VER social choice function is not implementable with verification. In particular, the example shows that, if there exists two valuations a and b and an outcome X such that $a(X) = b(X) \neq \bar{v}_i$ then there exist W-MON-VER social choice functions that are not implementable with verification.

5 One-Parameter Agents

In this section we present our results about *one-parameter* agents. As we shall see, one-parameter agents are a special case of agents with strongly comparable types, and thus Theorem 16 gives us a necessary and sufficient condition for a social choice function to be implementable with verification. In this section, the focus is on *efficiently* computable social choice functions (which will also be referred to as *algorithms*).

The main result of this section (see Theorem 34) shows, for a large class of optimization functions μ (see Definitions 29 and 33), how to transform a polynomial-time α-approximate algorithm for μ into an efficiently computable social choice function that is implementable with verification for one-parameter agents and $\alpha(1+\varepsilon)$-approximates μ. The class of function μ to which our transformation applies include several classical scheduling problems (see Section 6.1). In Theorem 36, we give a similar result for *online* settings.

Definition 18 *The valuation v_i of a* one-parameter agent *can be written as*

$$v_i(X) = -w_i(X) \cdot t_i,$$

for some publicly known non-negative function $w_i(\cdot)$ and some real number $t_i \geq 0$ that is privately known to agent i.

Observe that the valuation of a one-parameter agent is non-positive. We assume that when asked to report his type, an agents replies with a real number r_i, implying that he reports his valuation to be $b_i(X) = -w_i(X) \cdot r_i$. We consider optimization functions $\mu(X, b_1, \ldots, b_m)$ (as opposed to functions of the form $\mu(b_1(X), \ldots, b_m(X))$ of the previous section) that are non-decreasing in each valuation b_i and thus, equivalently, non-increasing in each reported type r_i. With some abuse of notation, we sometimes write $\mu(X, \mathbf{r})$ in place of $\mu(X, \mathbf{b})$, where $\mathbf{b}$ and $\mathbf{r}$ are related as above. Also, observe that one-parameter agents are a special case of strongly comparable types and, in particular, one has that $\bar{v}_i = 0$. One-parameter agents have been studied for scheduling optimization problems on *related* machines owned by selfish agents [1] (see Sect. 6).

In the rest of this section, we will show how to design social choice functions for one-parameter agents that are implementable with verification and that can be computed in polynomial time. By virtue of Theorem 16, it suffices to focus on social choice functions that are W-MON-VER for one-parameter agents. We first observe the following:

Fact 19 *For one-parameter agents, a social choice function A is W-MON-VER if and only if, for all i, $\mathbf{r}_{-i}$ there exists a* critical value $\theta_i \in (\Re^+ \cup \infty)$ *such that (i) $w_i(r_i, \mathbf{r}_{-i}) = 0$ for $r_i > \theta_i$, and (ii) $w_i(r_i, \mathbf{r}_{-i}) > 0$ for $r_i < \theta_i$.*

Proof. Let A be a W-MON-VER social choice function. Fix i and $\mathbf{r}_{-i}$. We show that if $w_i(r_i, \mathbf{r}_{-i}) = 0$ then for all $r_i' > r_i$ it holds $w_i(r_i', \mathbf{r}_{-i}) = 0$ and thus the minimum r_i for which $w_i(r_i, \mathbf{r}_{-i}) = 0$ is the desired θ_i. To reduce the cumbersome nature of our notation, we will suppress dependence on $\mathbf{r}_{-i}$. Let us assume, by contradiction, that $w_i(r_i') > 0$. We will show a 2-cycle in the verification-graph of negative weight thus contradicting the W-MON-VER condition of A. Given reported types $r_i < r_i'$ the respective valuations are $b_i = -w_i(r_i) \cdot r_i$ and $b_i' = -w_i(r_i') \cdot r_i'$, which implies $b_i > b_i'$. Since $w_i(r_i) = 0$ the verification-graph will have the 2-cycle $b_i \to b_i' \to b_i$ of weight:

$$r_i w_i(r_i') - r_i' w_i(r_i') = w_i(r_i')(r_i - r_i') < 0.$$

Conversely, we show that the existence of θ_i implies that, for all i and $\mathbf{b}_{-i}$, the verification-graph of A has no 2-cycles of negative weight. In $\mathcal{V}(\mathbf{b}_{-i})$ all valuations smaller than θ_i form a directed acyclic graph, while valuations larger than θ_i form a unique strongly connected component in which all edges have weight 0. This implies that there is no negative weight 2-cycle. □

Notice that with a slight abuse of notation we have denoted the critical value with θ_i even though it depends on i and $\mathbf{r}_{-i}$. The above property is called *weak monotonicity* in [3], and Theorem 16 implies one of the main results in that work.

The MAX operator. We are given a function μ and want to design a social choice function A that is implementable with verification (i.e., W-MON-VER) and, for a given vector $\mathbf{b}$ of declared types, returns an outcome X such that $\mu(X, \mathbf{b})$ is close to the maximum of μ over all choices of $X \in \mathcal{O}$. Moreover, we want A to be efficiently computable. A natural approach is to start from simple social choice functions and combine them together. Mu'Alem and Nisan [16] consider the following "MAX" operator:

$\mathbf{MAX}_\mu(A_1,A_2)$ operator
• compute $X_1 = A_1(\mathbf{b})$ and $X_2 = A_2(\mathbf{b})$;
• if $\mu(X_1, \mathbf{b}) \geq \mu(X_2, \mathbf{b})$ then return X_1 else return X_2.

For minimization problems, one can simply consider a 'MIN' operator defined as $\text{MIN}_\mu(A_1, A_2) := \text{MAX}_{-\mu}(A_1, A_2)$. Notice the slight abuse of notation in using MAX_μ both with social choice functions (as in the description of the MAX_μ operator) and outcomes (as in Theorem 8) as arguments.

In general, the fact that A_1 and A_2 are W-MON-VER does not guarantee that $\text{MAX}_\mu(A_1, A_2)$ is also W-MON-VER. We borrow (and adapt) the following definition from [16]:

Definition 20 *A social choice function A is* bitonic *w.r.t. $\mu(\cdot)$ if it is W-MON-VER and, for every i and $\mathbf{r}_{-i}$, one of the following two conditions holds for the function $g(x) := \mu(A(x, \mathbf{r}_{-i}), (x, \mathbf{r}_{-i}))$: (i) $g(x)$ is non-increasing for $x < \theta_i$ and non-decreasing for $x \geq \theta_i$; or (ii) $g(x)$ is non-increasing for $x \leq \theta_i$ and non-decreasing for $x > \theta_i$, where θ_i is the critical value.*

The following is the main technical contribution of this sections and will be used to prove Theorem 34.

Theorem 21 *If each A_i is bitonic w.r.t. $\mu(\cdot)$ then social choice function $MAX_\mu(A_1, A_2, \ldots, A_k)$:= $MAX_\mu(MAX_\mu(A_1, \ldots, A_{k-1}),A_k)$ is bitonic w.r.t. $\mu(\cdot)$. In this case, $MAX_\mu(A_1, A_2, \ldots, A_k)$ is W-MON-VER for one-parameter agents.*

The same results hold for the 'MIN' operator if each A_i is bitonic w.r.t. $-\mu(\cdot)$. Theorem 21 is proved by showing a connection between W-MON-VER social choice functions and monotone social choice functions for known single minded bidders (a special type of agents for combinatorial auctions studied in [16]). Details are given in the next section.

5.1 Known single minded bidders

In a combinatorial auction we have a set G of goods and the valuation v_i of agent i assigns a value $v_i(S)$ to each set $S \subseteq G$ of goods. An outcome $X = (X_1, \cdots, X_m)$ is an allocations of goods to the agents with agent i receiving set X_i.

Definition 22 ([16]) *Agent i is* known single-minded (KSM) *if for a publicly known set S_i of goods*

$$v_i(X) = \begin{cases} t_i & \text{if } X_i \supseteq S_i; \\ 0 & \text{otherwise.} \end{cases}$$

The value t_i is known to bidder i only.

Mu'Alem and Nisan characterized social choice functions that are implementable for KSM bidder.

Definition 23 *A social choice function A is* monotone for KSM bidders *if and only if, for all i, $\mathbf{r}_{-i}$ there exists a* critical value *$\theta_i \in (\Re^+ \cup \infty)$ such that (i) any $r_i > \theta_i$ is a winning declaration, and (ii) any $r_i < \theta_i$ is a non-winning declaration.*

We have the following theorem.

Theorem 24 *[16] A social choice function A is implementable for KSM bidders if and only if A is monotone.*

The next result provides an interesting connection between monotone algorithms for KSM bidders and W-MON-VER algorithms for one-parameter agents. This connection is made by considering that, in both cases, an algorithm receives in input m numbers $r_1, \ldots, r_m$, corresponding to the agents declarations.

Theorem 25 *Let $\mathcal{AG}$ be a set of m one-parameter agents, where agent i has valuation $v_i = -w_i(X) \cdot t_i$. Also let $\mathcal{AG}'$ be a set of KSM bidders with valuation*

$$v'_i(X) := \begin{cases} t_i & \text{if } w_i(X) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Then, social choice function A is W-MON-VER w.r.t. the one-parameter agents if and only if A is monotone w.r.t. the KSM bidders.

PROOF. We stress that A takes in input the parameters r_i in both cases. For every i and $\mathbf{r}_{-i}$, a value r_i is a winning declaration if and only if $w_i(A(r_i, \mathbf{r}_{-i})) = 0$. By definition of monotone social choice-function (i) $\forall\ r_i > \theta_i$, $w_i(A(r_i, \mathbf{r}_{-i})) = 0$ and (ii) $\forall\ r_i < \theta_i$, $w_i(A(r_i, \mathbf{r}_{-i})) > 0$. The theorem follows from Fact 19. □

We will use this result to import several of the results about KSM bidders in [16]. In the sequel, we will use the terms "W-MON-VER" and "monotone for KSM bidders" interchangeably. In particular, Def. 20 is equivalent to the following one:

Definition 26 ([16]) *An algorithm A is* bitonic *w.r.t. $\mu(\cdot)$ if it is monotone (for KSM bidders) and, for every i and $\mathbf{r}_{-i}$, there exists a threshold θ_i such that one of the following two conditions holds for the function $g(x) := \mu(A(x, \mathbf{r}_{-i}), (x, \mathbf{r}_{-i}))$: (i) $g(x)$ is non-increasing for $x < \theta_i$ and non-decreasing for $x \geq \theta_i$; or (ii) $g(x)$ is non-increasing for $x \leq \theta_i$ and non-decreasing for $x > \theta_i$. (The value θ_i is defined as in Definition 23.)*

The following result has been proved in [16] for the case $\mu(\cdot)$ is the social welfare, i.e., the sum of all agents valuations $v_i(S)$. However, as already observed in [5], the proofs do not make use of this assumption and all the results hold when replacing the social welfare with any function $\mu(\cdot)$.

Theorem 27 ([16]) *If A_1 and A_2 are bitonic w.r.t. $\mu(\cdot)$, then $A = MAX_\mu(A_1, A_2)$ is bitonic w.r.t. $\mu(\cdot)$ as well.*

Because of the above result, we can apply the "MAX" operator to more than two algorithms. Hence, $\mathrm{MAX}(A_1, \ldots, A_k)$ is bitonic if each algorithm A_i is bitonic. By Theorem 25, it follows that:

Theorem 28 *Let $A_1, \ldots, A_k$ be algorithms satisfying Def. 20. Then algorithm $MAX_\mu(A_1, \ldots, A_k)$ satisfies Def. 20 as well.*

Hence, Theorem 21 follows.

5.2 Efficient W-MON-VER social choice functions

Theorem 21 provides a powerful tool for efficiently building social choice functions starting from simpler ones. In particular, we will use this result to extend Theorem 8 to a wider class of optimization functions of the form $\mu(X, b_1, \ldots, b_m)$. This allows us to deal with certain scheduling problems where the measure depends on the scheduling policy internal to the machines and therefore cannot be expressed as the machines completion times (i.e., as a function of $w_i(X) \cdot t_i$).

We start by the defining the notion of neutral functions.

Definition 29 *A function $\mu(\cdot)$ is* neutral *if, for every X such that $w_i(X) = 0$, it holds that $\mu(X, (b_i, \mathbf{b}_{-i})) = \mu(X, (b'_i, \mathbf{b}_{-i}))$, for every b_i, b'_i and every $\mathbf{b}_{-i}$.*

We have the following technical lemma.

Lemma 30 *Let $\mu(X, b_1, \ldots, b_m)$ be neutral and non-decreasing in each b_i, for every $X \in \mathcal{O}$. Then any algorithm returning a fixed outcome X is bitonic w.r.t. $\mu(\cdot)$.*

PROOF. Since X is fixed then we have two possible cases: either $w_i(X) > 0$ or $w_i(X) = 0$. In the first case the critical value is set to $\theta_i = \infty$. Since $\mu(\cdot)$ is non-decreasing in b_i, then $\mu(\cdot)$ is non-increasing in r_i and therefore $g(x) := \mu(X, (x, \mathbf{r}_{-i}))$ is non-increasing for $x < \theta_i$. In the second case the critical value is set to $\theta_i = 0$. Since, $w_i(X) = 0$ and $\mu(\cdot)$ is neutral, $g(x)$ is constant and thus non-decreasing for $x \geq \theta_i$. □

Theorem 31 *Let $\mu(X, b_1, \ldots, b_m)$ be neutral and non-decreasing in each b_i, for every $X \in \mathcal{O}$. Then, for any $X_1, \ldots, X_\ell \in \mathcal{O}$, the social choice function $A = MAX_\mu(X_1, \ldots, X_\ell)$ is bitonic w.r.t. $\mu(\cdot)$. Hence A is implementable with verification.*

PROOF. By Lemma 30, algorithms returning one fixed outcome are bitonic. Therefore, the theorem follows from Theorem 21. □

Theorem 31 above has two important consequences. First of all, we can obtain a result (similar to Theorem 9) that shows that optimization of neutral monotone function $\mu(X, b_1, \ldots, b_m)$ for one parameter agents can be implemented with verification.

Theorem 32 *For one-parameter agents and for any function $\mu(X, b_1, \ldots, b_m)$ which is neutral and non-decreasing in each b_i, there exists a social choice function OPT_μ that maximizes $\mu(\cdot)$ and is implementable with verification.*

Another consequence is that, if we have an α-approximation preserving set of outcomes $\mathcal{O}'$ for μ, we can apply the above theorem to all outcomes $X \in \mathcal{O}'$. This gives us a social choice function A which is implementable with verification, α-approximates μ and can be computed in time polynomial in $|\mathcal{O}'|$.

We next introduce the class of smooth functions, for which there exists small α-approximation preserving set of outcomes.

Definition 33 *Fix $\varepsilon > 0$ and $\gamma > 1$. A function μ is (γ, ε)–*smooth *if, for any pair of declarations $\mathbf{r}$ and $\tilde{\mathbf{r}}$ such that $r_i \leq \tilde{r}_i \leq \gamma r_i$ for $i = 1, 2, \ldots, m$, and for all possible outcomes X, it holds that $\mu(X, \mathbf{r}) \leq \mu(X, \tilde{\mathbf{r}}) \leq (1 + \varepsilon) \cdot \mu(X, \mathbf{r})$.*

For smooth, neutral functions μ we can transform any α-approximate polynomial-time algorithm A (which is not necessarily implementable with verification) into a social choice function for a constant number of agents which is computable in polynomial-time, implementable with verification and $\alpha(1+\varepsilon)$-approximates μ.

Theorem 34 *Let A be a polynomial-time α-approximate algorithm for a neutral, non-decreasing (in each b_i) (γ, ε)-smooth objective function $\mu(\cdot)$. Then, for any $\varepsilon > 0$, there exists an $\alpha(1 + \varepsilon)$-approximate social choice function $A^\star$ implementable with verification. If the number of agents is constant, $A^\star$ can be computed in polynomial time.*

PROOF SKETCH. Let $\mathcal{O}'$ be the set of outcomes returned by A when run on bid vectors whose components are powers of γ. For m agents, $|\mathcal{O}'| = O(\max_i\{\log_\gamma |D_i|\}^m)$ which is polynomial for fixed m, and, since μ is (γ, ε)-smooth, $\mathcal{O}'$ is an $\alpha(1 + \varepsilon)-$approximation preserving set for μ.

Consider social choice function $A^\star$ that on input $\mathbf{r}$ outputs the outcome $X \in \mathcal{O}'$ that maximizes $\mu(X, \mathbf{r})$. By Theorem 31, $A^\star$ is W-MON-VER and $\alpha(1 + \varepsilon)$-approximates μ. Moreover, for constant m, $A^\star$ is polynomial-time computable. □

5.3 Online mechanism

A natural way of designing an *online* algorithm for scheduling problems is to iterate a "basic-step" algorithm B which, given the current assignment X, the processing requirement of the new job J and the reported types $b_1, \ldots, b_m$ (that is, the reported speed of machine i is $1/r_i$) outputs the index $B(X, J, \mathbf{b})$ of the machine to which the job must be assigned. For algorithm B, the set of outcomes $\mathcal{O}$ consists of all allocations that can be obtained from X by allocating job J to one of the m machines.

Algorithm B-iterated$(\mathbf{b})$
- $X := \emptyset$;
- while a new job J arrives do
- assign job J to machine of index $B(X, J, \mathbf{b})$ and modify X accordingly.

Observe that a basic-step algorithm B that is implementable with verification does not necessarily remains implementable with verification when iterated. Also in this case stability plays a central role.

We have the following theorem.

Theorem 35 *If B is stable then algorithm B-iterated is stable as well.*

PROOF. We prove the theorem inductively and show that, if B-iterated is stable for the first k jobs, then it remains stable after the $(k+1)$-th job J_{k+1} is allocated. Let $\mathbf{J}^k$ be the sequence of the first k jobs. Let $\mathbf{b} = (b_i, \mathbf{b}_{-i})$ and $\mathbf{b}' = (b'_i, \mathbf{b}_{-i})$ with $b_i > b'_i$ and let $Y = B\text{-iterated}(\mathbf{b}, \mathbf{J}^{k+1})$. We prove that if $w_i(Y) = 0$ then $Y' = B\text{-iterated}(\mathbf{b}', \mathbf{J}^{k+1}) = Y$. Since $w_i(Y) = 0$ then $w_i(X) = 0$ for $X = B\text{-iterated}(\mathbf{b}, \mathbf{J}^k)$. By inductive hypothesis, B-iterated is stable for $\mathbf{J}^k$, thus implying $B\text{-iterated}(\mathbf{b}', \mathbf{J}^k) = X$. Since B is stable and $w_i(X) = 0$ it holds $B(X, \mathbf{b}, J_{k+1}) = B(X, \mathbf{b}', J_{k+1})$. By definition of B-iterated we obtain that $B\text{-iterated}(\mathbf{b}', \mathbf{J}^{k+1}) = B\text{-iterated}(\mathbf{b}, \mathbf{J}^{k+1})$. □

Therefore, by Theorem 7, B-iterated is implementable with verification. For example, Graham's [11] online greedy algorithm for $Q||C_{\max}$ can be seen as the iterated version of a simple basic stable step and thus it is implementable with verification. This property holds more in general. Consider the *greedy* algorithm which, at every step, assigns a newly arrived job to the machine that, given the current assignment of previous jobs, maximizes the increase of the objective function $\mu(\cdot)$; ties are broken in a fixed manner, and $-\mu(\cdot)$ is typically a cost function that one wishes to minimize (e.g., the L_p norm defined as $\sqrt[p]{\sum_i (w_i(X) \cdot t_i)^p}$). We have the following theorem.

Theorem 36 *The greedy algorithm is stable for cost functions $\mu(X, b_1, \ldots, b_m)$ that are neutral and non-decreasing in each b_i, for every $X \in \mathcal{O}$.*

PROOF. Because of Theorem 35 we only need to prove that the basic-step B of the greedy algorithm is stable. Consider two bid vectors $\mathbf{b} = (b_i, \mathbf{b}_{-i})$ and $\mathbf{b}' = (b'_i, \mathbf{b}_{-i})$ with $b'_i < b_i$ (and thus $r'_i > r_i$) and let X be the schedule computed by the greedy algorithm w.r.t. both bid vectors for the first k jobs and assume that X is such that $w_i(X) = 0$. Now suppose that, for the $(k+1)$-th job, greedy computes a solution Y (with $w_i(Y) = 0$) on input $\mathbf{b}$ and, by contradiction, assume that on input $\mathbf{b}'$ greedy computes a solution $Y' \neq Y$. Observe that

$$\begin{array}{lcll} \mu(Y', \mathbf{b}') & \geq & \text{(by the definition of greedy algorithm)} & (14) \\ \mu(Y, \mathbf{b}') & = & \text{(by } w_i(Y) = 0 \text{ and by neutrality of } \mu) & (15) \\ \mu(Y, \mathbf{b}) & \geq & \text{(by the definition of greedy algorithm)} & (16) \\ \mu(Y', \mathbf{b}) & \geq & \text{(since } \mu \text{ is non-decreasing in each } b_i) & (17) \\ \mu(Y', \mathbf{b}'). & & & (18) \end{array}$$

Let B be the basic-step of algorithm greedy. Then solution Y is obtained from a solution X by allocating the $(k+1)$-th job to machine of index $p = B(X, \mathbf{b}, J)$. Similarly, solution Y' is obtained from the solution X by allocating the $(k+1)$-th job to machine of index $q = B(X, \mathbf{b}, J)$. The above inequalities imply that $\mu(Y, \mathbf{b}) = \mu(Y', \mathbf{b}')$. Since B breaks ties in a fixed order, it must be $p = q$, thus contradiction the hypothesis $Y \neq Y'$. □

Hence, if the greedy algorithm is α-approximating for such a function $\mu(\cdot)$, then there exists an online polynomial-time computable algorithm implementable with verification and that is an α-approximation for $\mu(\cdot)$.

6 Applications

In this section we discuss two types of applications of our results for one-parameter agents: polynomial-time approximation mechanisms with verification for scheduling problems and exact mechanisms *without* verification for certain graph problems.

6.1 Scheduling on Related Machines

We consider scheduling problems on related machines owned by selfish agents as in [1]. We are given a set of m *related machines* and a set of n jobs. Assigning a job to machine i makes the work w_i of that machine to increase by an amount equal to the job weight (each job has a weight and a job can be assigned to any machine). Each machine i has a *speed* s_i, that is, the completion time of machine i is w_i/s_i, where w_i is the *work* assigned to machine i (i.e., the sum of all job weights that are assigned to that machine). Thus, the set of jobs consists of a vector $\mathbf{J} = (J_1, J_2, \ldots, J_n)$, where J_k is the weight of

the k-th job. In the *online* setting, jobs arrive one-by-one, the k-th job must be scheduled before next one arrives, and jobs cannot be reallocated.

For an assignment X, we let $w_i(X)$ be the work that this solution assigns to machine i. Each machine i corresponds to a *selfish agent* whose valuation is $-w_i(X)/s_i = -w_i(X) \cdot t_i$ for some $t_i = 1/s_i$; i.e., the speed of machine i is known to agent i only (everything else is known to the mechanism) and her valuation is the opposite of the completion time of her machine. An agent can thus misreport her speed (i.e., declare $r_i \neq t_i$).

Mechanisms with verification compute, for each agent i, an associated payment and award agent i her payment if and only if all jobs assigned to machine i have been released after $w_i(X) \cdot r_i$, where X is the outcome selected by the mechanism [18, 3]. Machine i can misreport her speed and still receive her associated payment if one of the following happens: (i) the declared speed is worse (i.e., $r_i < t_i$) and jobs are released accordingly by adding some delay; (ii) the declared speed is better (i.e., $r_i < t_i$) but this makes the allocation algorithm A to compute an allocation X which does not assign any job to machine i (i.e., $w_i(X) = 0$, in which case no verification is possible).

We consider several variants of this problem depending on the optimization function adopted. All of these problems are *minimization problems* for which it is NP-hard to compute exact solutions, even for $m = 2$. The following table summarizes some of the applications of our techniques to scheduling problems.

Problem version	**Upper Bound**	
	Exp Time (any m)	**Polytime (m constant)**
$Q\|\|\sum_j \mathrm{w_j C_j}$	OPT [3]	$(1+\varepsilon)$-**approximate [Thm. 34 & [6]]**
$Q\|\mathrm{r_j}\|\sum_j \mathrm{w_j C_j}$	**OPT [Thm. 32]**	$(1+\varepsilon)$-**approximate [Thm. 34 & [6]]**
$Q\|prec, r_j\|\sum_j \mathrm{w_j C_j}$	**OPT [Thm. 32]**	$O(\log m)$-**approximate [Thm. 34 & [7]]**
L_p norm	**OPT [Thm. 32]**	$O(p)$-**competitive [Thm. 36]**

For the first three problems, no mechanism without verification can attain an approximation factor better than $2/\sqrt{3} > 1$, for all $m \geq 2$ [1]. Our upper bounds (in bold) are the first bounds on these problems which are all NP-hard solve exactly; bounds for $Q||\sum_j \mathrm{w_j C_j}$ and $Q|\mathrm{r_j}|\sum_j \mathrm{w_j C_j}$ break the $2/\sqrt{3}$ lower bound in [1], which also holds for exponential-time mechanisms; upper bound for the L_p norm is obtained via online mechanisms based on the greedy algorithm (for $p = 2$, the bound is $1 + \sqrt{2}$).

Weighted Sum Scheduling ($Q||\sum_j \mathrm{w_j C_j}$). Each job j has also a *priority* $\mathrm{w_j} \in \Re^+$ and the goal is to minimize $\sum_j \mathrm{w_j C_j}$, where $\mathrm{C_j}$ is the completion time of job j. The set of outcomes $\mathcal{O} = \mathcal{O}(\mathbf{J})$ consists of all possible allocations of jobs to machines. Given a job assignment $X \in \mathcal{O}$, every machine processes jobs from the smallest to the largest and this determines the completion time $\mathrm{C_j}(X, r_1, \ldots r_m)$ of job j. [1] We can thus consider a function

$$\mu(X, r_1, \ldots, r_m) := -\sum \mathrm{w_j C_j}(X, r_1, \ldots, r_m),$$

where $\mathrm{C_j}(X, r_1, \ldots, r_m)$ is non-decreasing in each r_i. Indeed, if X assigns job j to machine $X(j)$ and a $prec_j(X)$ is the set of jobs that, according to X, are scheduled on the same machine before j, then

$$\mathrm{C_j}(X, r_1, \ldots, r_m) := \sum_{j' \in prec_j(X) \cup j} J_{j'} \cdot r_{X(j)}.$$

Since $\mathrm{C_j}(X, r_1, \ldots, r_m)$ is non-decreasing in each r_i, then $\mu(X, r_1, \ldots, r_m)$ is non-increasing in each r_i. Recall that, according to our terminology, b_i and r_i are related by $b_i(X) = -w_i(X) \cdot r_i$. In this case, we write $\mu(X, r_1, \ldots, r_m)$ simply as $\mu(X, \mathbf{b})$, where $\mathbf{b} = (b_1, \ldots, b_m)$. Hence, $\mu(X, \mathbf{b})$ is non-decreasing in each b_i. Clearly $\mu(\cdot)$ is also neutral and (γ, ε)–smooth for every $\varepsilon > 0$. Therefore, Theorem 34 and

[1] Although other policies are possible, it is well-known that this policy minimizes the optimization function that we are going to consider in this problem. Moreover, changing the scheduling policy internal to a machine does not affect the completion time of that machine, and thus the agent's valuation.

the $(1+\varepsilon)$–approximation scheme in [6] imply the existence of a polynomial-time $(1+\varepsilon)$–approximation mechanism with verification, for any $m = O(1)$ and any $\varepsilon > 0$.

This problem version has been first considered in [1], where the authors proved that no mechanism without verification can attain an approximation factor better than $2/\sqrt{3}$, even for the case $m = 2$ and using mechanisms which do not run in polynomial time.

Weighted Sum Scheduling with Release Times ($Q|\mathrm{r_j}|\sum_j \mathrm{w_j C_j}$) and Precedence Constraints ($Q|prec_j, \mathrm{r_j}|\sum_j \mathrm{w_j C_j}$). In the $Q|\mathrm{r_j}|\sum_j \mathrm{w_j C_j}$ problem each job has also a *release* time, that is, job j cannot be scheduled on a machine before time $\mathrm{r_j}$. This affects the feasible allocation of jobs and the scheduling policies internal to the machines (i.e., the order in which a machine processes its jobs). Hence, the only difference with the previous problem is on the set of possible outcomes $\mathcal{O}$ which depends on the job weights $\mathbf{J}$ and on each $\mathrm{r_j}$ (this part of the input is known to the mechanism). In [6] the authors give a $(1+\varepsilon)$–approximation scheme which, as in the previous example, can be turned into a polynomial-time $(1+\varepsilon)$–approximation mechanism with verification, for any $m = O(1)$ and any $\varepsilon > 0$.

In the $Q|prec_j, \mathrm{r_j}|\sum_j \mathrm{w_j C_j}$ we are also given precedence constraints $prec_j$ for each job which limits the order of executions among jobs. In this case the set of feasible outcomes depends on $\mathbf{J}$, on every $\mathrm{r_j}$ and on every $prec_j$. In [7] the authors gave an $O(\log m)$-approximation algorithm which, as above, can be turned into a polynomial-time $O(\log m)$–approximation mechanism with verification, for any $m = O(1)$. [2]

Makespan and Norm L_p. Possible outcomes consists of all job allocations to the machines. In the L_p norm, given the machine speeds $s_1, \ldots, s_m$ and a job allocation X, the objective function that one wishes to minimize is

$$\left(\sum_{i=1}^{m} (w_i(X)/s_i)^p \right)^{1/p}.$$

We can thus consider maximizing the function

$$\mu(X, b_1, \ldots, b_m) := -\left(\sum_{i=1}^{m} (w_i(X)/s_i)^p \right)^{1/p} = \left(\sum_{i=1}^{m} (b_i(X))^p \right)^{1/p}.$$

The L_∞ norm corresponds to minimizing the *makespan*, that is, $\max_{i=1}^{m}\{-w_i(X)/s_i\}$. In both L_p and L_∞ norm, the corresponding function $\mu(\cdot)$ is clearly neutral, non-decreasing in b_i, and (γ, ε)–smooth for every $\varepsilon > 0$.

Interestingly, for the L_p norm the greedy algorithm guarantees a constant approximation (and thus a constant competitive ratio) [4]. For $p = 2$, the bound is $(1+\sqrt{2})$. Hence, Theorem 36 implies the following:

Corollary 37 *For the L_p norm, there exists an online $O(p)$-approximate mechanism with verification. For $p > 2$ the approximation factor is $(1+\sqrt{2})$.*

6.2 Graph Problems

Consider problems in which we are given a graph $G = (V, E)$ and the set of feasible outcomes consists of a suitable set $\mathcal{O} = \mathcal{O}(G)$ containing certain subgraphs of G. We have one agent per edge and the type t_e of agent e is nothing but the *weight* of edge $e \in E$. If subgraph $X \in \mathcal{O}$ of G includes edge e, then agent e has a cost (for implementing this outcome) equal to t_e. This scenario is common to several problems considered in the algorithmic mechanism design community: shortest-path [18], minimum spanning tree [18], minimum-radius spanning tree [21].

We next show how to obtain exact mechanisms *without* verification for such graph problems:

[2] Although we consider $m = O(1)$, we write $O(\log m)$ to stress that the (constant) in the approximation factor grows as the logarithm of the number of machines.

Definition 38 *A* mechanism design graph problem *consists of a set of outcomes $\mathcal{O}(G)$ depending on a graph $G = (V, E)$. Each $X \in \mathcal{O}(G)$ consists of a suitable subgraph of G. There is one agent per edge and the valuation of agent e is*

$$v_e(X) := \begin{cases} -t_e & \text{if } e \in X, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that, in mechanism design graph problems, we deal with a special case of one-parameter agents in which $w_e(X) = 1$ if edge e is used by X, and $w_e(X) = 0$ otherwise. Hence, a social choice function A is W-MON-VER if and only if A is monotone according to the following definition:

Definition 39 (monotone algorithm) *A social choice function A is* monotone *(for one-parameter agents) if, for all agents e, and for all r_e, $\mathbf{r}_{-e}$ and $r'_e < r_e$, it holds that $w_e(A(r_e, \mathbf{r}_{-e})) \leq w_e(A(r'_e, \mathbf{r}_{-e}))$.*

We make use of the following result:

Theorem 40 ([17, 1]) *For any A monotone (for one-parameter agents), there exists P such that $M = (A, P)$ is a truthful mechanism (without verification).*

We thus obtain the following general result which states that our results for one-parameter agents can be used to derive mechanisms which do not make use of verification:

Theorem 41 *Let A be any W-MON-VER social choice function for a mechanism design graph problem. Then there exists payment functions P such that (A, P) is a truthful mechanism without verification.*

PROOF. Since $w_e(X) \in \{0, 1\}$, Fact 19 implies that A is W-MON-VER if and only if A is monotone according to Def. 39. Theorem 40 thus implies the existence of the payments. □

Corollary 42 *Let $A_1, \ldots, A_k$ be k algorithms for a mechanism design graph problem, each of them being bitonic w.r.t $\mu(\cdot)$. Then, there exists payment functions P such that (A, P) is a truthful mechanism without verification. The same result holds for the 'MIN' operator if each A_i is bitonic w.r.t. $-\mu(\cdot)$. Hence, for mechanisms design graph problems with a minimization function $\mu(\cdot)$ which is neutral and monotone non-decreasing in each r_i, there exists an exact truthful mechanism without verification.*

We next apply this result to the following *non-utilitarian* problem:

Minimum Radius Spanning Tree [21]. The set of feasible outcomes $\mathcal{O}(G)$ consists of all trees over a given graph $G = (V, E)$. The goal is to find a rooted tree of minimum height with respect to the edge weights t_e, $e \in E$. That is, a tree T_u rooted at some $u \in V$ such that the longest path from any v to the root u is as small as possible (the length of a path is the sum of all edge weights t_e of edges e in that path). As observed by the authors of [21], this problem is *not* utilitarian and some algorithms which solve this problem exactly cannot lead to truthful mechanisms. The authors the provide a polynomial-time mechanism by sticking to a particular way of computing the optimal solutions which breaks ties in a fixed manner: let SPT_u be the algorithm computing a shortest-path tree of G rooted at $u \in V$; then the minimum-radius spanning tree can be obtained by selecting the best, over all $u \in V$, of the solution returned by SPT_u. In other words, the following algorithm solves this problem (see [21] for the details):

$$MIN_{height}(SPT_{u_1}, \ldots, SPT_{u_{|V|}}),$$

where $u_i \in V$, for $1 \leq i \leq |V|$, and $height(\cdot)$ is the height of the corresponding tree. One can show that each algorithm SPT_u is bitonic with respect to $height(\cdot)$: as long as SPT_u uses edge e, increasing its weight r_e cannot decrease the height of the solution; as soon as SPT_u drops this edge, then r_e does not affect the height of that solution. Thus, the function '$-g(x)$' in Definition 20 thus looks like a "stairway" which becomes constant when edge e is dropped (see also [21] for the details). Corollary 42 thus gives an alternative proof of the following:

Theorem 43 ([21]) *The minimum radius spanning tree problem admits an exact truthful mechanism (without verification). The underlying algorithm runs in polynomial-time.*

Besides the application to the minimum-radius spanning tree problem, Corollary 42 provides a powerful tool for proving that certain problems admit exact truthful mechanisms without verification.

Remark 44 *The work [21] also provides polynomial-time computable payments based on the payment functions in [1]. This step is non-trivial and, in general, it would be interesting to obtain a general result in this direction. That is, if it is possible to (efficiently) compute the payments for 'MAX' or 'MIN'.*

References

[1] Aaron Archer and Eva Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of the Annual IEEE Symposium on Foundations of Computing (FOCS)*, pages 482–491, 2001.

[2] Vincenzo Auletta, Roberto De Prisco, Paolo Penna, and Giuseppe Persiano. On designing truthful mechanisms for online scheduling. In *Proc. of the 12th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 3499 of *Lecture Notes in Computer Science*, pages 3–17. Springer Verlag, 2005.

[3] Vincezo Auletta, Roberto De Prisco, Paolo Penna, and Giuseppe Persiano. The power of verification for one-parameter agents. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 3142 of *LNCS*, pages 171–182, 2004.

[4] Baruch Awerbuch, Yossi Azar, Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. Load balancing in the L_p norm. In *Proc. of Annual IEEE Symposium on Foundations of Computing (FOCS)*, pages 383–391, 1995.

[5] Patrick Briest, Piotr Krysta, and Berthold Vöcking. Approximation techniques for utilitarian mechanims design. In *Proc. of 37th Annual ACM Symposium on Theory of Computing (STOC)*, ACM Press, pages 39–48, 2005.

[6] Chandra Chekuri and Sanjeev Khanna. A ptas for minimizing weighted completion time on uniformly related machines. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 2076 of *LNCS*, 2001.

[7] Fabian A. Chudak and David B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

[8] Edward H. Clarke. Multipart Pricing of Public Goods. *Public Choice*, pages 17–33, 1971.

[9] S. Dobzinski, Noam Nisan, and M. Schapira. Approximation Algorithms for Combinatorial Auctions with Complement-Free Bidders. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 610–618, 2005.

[10] Shahar Dobzinski, Noam Nisan, and Michael Schapira. Truthful randomized mechanisms for combinatorial auctions. In *To appear in Proceedings of the 38th Symposium on Theory of Computing (STOC06)*, 2006.

[11] Ron L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[12] Theodore Groves. Incentive in Teams. *Econometrica*, 41:617–631, 1973.

[13] Hongwei Gui, Rudolf Muller, and Rakesh V. Vohra. Dominant strategy mechanisms with multidimensional types. Technical report, 2004.

[14] Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, Mohammad Mahdian, and David C. Parkes. Online auctions with re-usable goods. In *Proceedings 6th ACM Conference on Electronic Commerce (EC-2005)*, pages 165–174. ACM, 2005.

[15] Ron Lavi, Ahuva Mu'Alem, and Noam Nisan. Towards a characterization of truthful combinatorial auctions. In *In Proceedings of Annual IEEE Symposium on Foundations of Computing (FOCS)*, 2003.

[16] Ahuva Mu'Alem and Noam Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. In *Proc. of 18th National Conference on Artificial intelligence (AAAI)*, pages 379–384, 2002.

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

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

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

[20] Ryan Porter. Mechanism design for online real-time scheduling. In *Proceedings of the 5th ACM Conference on Electronic Commerce (EC-2004)*, pages 61–70. ACM, 2004.

[21] Guido Proietti and Peter Widmayer. A truthful mechanism for the non-utilitarian minimum radius spanning tree problem. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, ACM Press, pages 195–202, 2005.

[22] Kevin Roberts. The Characterization of Implementable Choice Rule. In J. J. Laffont, editor, *Aggregation and Revelation of Preferences*, 1979.

[23] J.-C. Rochet. A condition for rationalizability in a quasi-linear context. *Journal of Mathematical Economics*, 16:191–200, 1987.

[24] Michael Saks and Lan Yu. Weak monotonicity suffices for truthfulness on convex domains. In *EC '05: Proceedings of the 6th ACM conference on Electronic commerce*, pages 286–293, New York, NY, USA, 2005. ACM Press.

[25] William Vickrey. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance*, pages 8–37, 1961.

A Comparable Type Sets for Multidimensional Agents

In this section we show that extending our results for multidimensional agents does not seem to be immediate. We only discuss the case of bi-dimensional agents. The type $\mathbf{v} = (v_1, v_2)$ of a bi-dimensional comparable agent has two components each of which is a one-dimensional comparable valuation and a solution X also consists of two components. A *comparable bi-dimensional type sets* is simply the cross product of two comparable (unidimensional) type sets and we will denote by $D_i = D_i^1 \times D_i^2$ the type set of agent i. The valuation $\mathbf{v}$ for a given solution X is a function of v_1 and v_2 (e.g., $\mathbf{v}(X) = v_1(X) + v_2(X)$). We assume that the mechanism is able to verify each of the two coordinates independently. This implies that if an agent is caught lying over one of the two components (for example, the agent declared b_1 as first component of its type instead of v_1 and the solution computed by the mechanisms X is such that $b_1(X) > v_1(X)$) then the agent receives no payment.

Definition 45 (2D verification graph) *Fix agent i and $\mathbf{b}_{-i}$. The* 2D verification graph $\mathcal{V}(\mathbf{b}_{-i})$ *of agent i with respect to social choice function A and reported types $\mathbf{b_{-i}}$ of the other agents has a node for each element of D_i.*

The edges of $\mathcal{V}(\mathbf{b}_{-i})$ are of two different kinds. Let $\mathbf{a} = (a_1, a_2), \mathbf{b} = (b_1, b_2) \in D_i$ be two types.

Natural edges. *If $b_1 \leq a_1$ and $b_2 \leq a_2$, the graph $\mathcal{V}(\mathbf{b}_{-i})$ contains the edge $(\mathbf{a}, \mathbf{b})$.*

Unnatural edges. *The graph $\mathcal{V}(\mathbf{b}_{-i})$ contains the edge $(\mathbf{a}, \mathbf{b})$ if (i) $b_1 \leq a_1$, $a_2 \leq b_2$ and $a_2(A_{\mathbf{b}_{-i}}(\mathbf{b})) = b_2(A_{\mathbf{b}_{-i}}(\mathbf{b}))$, or (ii) $a_1 \leq b_1$, $b_2 \leq a_2$ and $a_1(A_{\mathbf{b}_{-i}}(\mathbf{b})) = b_1(A_{\mathbf{b}_{-i}}(\mathbf{b}))$.*

The edge $(\mathbf{a}, \mathbf{b})$, if it exists in $\mathcal{V}(\mathbf{b}_{-i})$, has weight $\mathbf{a}(A_{\mathbf{b}_{-i}}(\mathbf{a})) - \mathbf{a}(A_{\mathbf{b}_{-i}}(\mathbf{b}))$.

Specifically, for each type $\mathbf{a} \in D_i$, $\mathcal{V}(\mathbf{b}_{-i})$ contains the edges $(\mathbf{a}, \mathbf{b})$ for all $\mathbf{b} \in D_i$ for which agent i (with type $\mathbf{a}$) can report type $\mathbf{b}$ without being caught by the verification. This can happen because of one of the following reasons: each component of $\mathbf{a}$ is no smaller than the corresponding component of $\mathbf{b}$ (in which case we have a natural edge); one component of $\mathbf{a}$, say the first one, is no smaller than the corresponding component of $\mathbf{b}$, the second component of $\mathbf{a}$ is no greater than the second component of $\mathbf{b}$ but the two valuations agree on the solution computed for reported types $(\mathbf{b}_{-i}, \mathbf{b})$.

As before, it is necessary, in order to obtain a truthful mechanism with verification, that the social choice function is W-MON-VER. In the one-dimensional case we have shown that the *stability* of the social choice function is sufficient. The following definition generalizes stability for bi-dimensional types.

Definition 46 (2D-stable) *A social function A is* 2D-stable *if, for all i, for all $\mathbf{b}_{-i}$, and for all unnatural edges $(\mathbf{a}, \mathbf{b})$, it holds that $a_1(A_{\mathbf{b}_{-i}}(\mathbf{b})) = b_1(A_{\mathbf{b}_{-i}}(\mathbf{b}))$ and $a_2(A_{\mathbf{b}_{-i}}(\mathbf{b})) = b_2(A_{\mathbf{b}_{-i}}(\mathbf{b}))$ implies $A_{\mathbf{b}_{-i}}(\mathbf{a}) = A_{\mathbf{b}_{-i}}(\mathbf{b})$.*

An easy generalization of Th. 8 shows that the social choice function that returns the lexicographically minimum solution that achieves the maximum of a monotone non-decreasing function μ is 2D-stable.

Unfortunately, we next give an example of a 2D-stable social choice function that is not implementable with verification.

Theorem 47 *There exists a 2D-stable social choice function A that is not implementable with verification.*

Proof. Consider types $\mathbf{a} = (a_1, a_2)$, $\mathbf{b} = (b_1, b_2)$ and $\mathbf{c} = (b_1, c_2)$ with $a_1 < b_1$ and $b_2 < c_2 < a_2$. Fix i and $\mathbf{b}_{-i}$ and set $X = A_{\mathbf{b}_{-i}}(\mathbf{a})$, $Y = A_{\mathbf{b}_{-i}}(\mathbf{b})$ and $Z = A_{\mathbf{b}_{-i}}(\mathbf{c})$. Also assume that

$$
\begin{aligned}
b_2(Z) &= c_2(Z), && (19)\\
a_1(Y) &= b_1(Y), && (20)\\
c_2(X) &= a_2(X), && (21)\\
a_1(X) &< b_1(X), && (22)\\
X &\neq Y. && (23)
\end{aligned}
$$

Equations 19-21 imply that $\mathcal{V}(\mathbf{b}_{-i})$ contains the three (unnatural) edges $(\mathbf{a},\mathbf{b})$, $(\mathbf{b},\mathbf{c})$ and $(\mathbf{c},\mathbf{a})$. From Eq. 19 and from the fact that A is 2D-stable we obtain that $Y = Z$. Thus the weight of the edge $(\mathbf{b},\mathbf{c})$ is 0. Now we compute the weight of the cycle $\mathbf{a} \to \mathbf{b} \to \mathbf{c} \to \mathbf{a}$ in the case in which the bi-dimensional valuation is the sum of the one-dimensional valuations. From Eq.s 20-23 and by observing that $c_2 \leq a_2$, we have:

$$a_2(X) + a_1(X) - a_2(Y) - a_1(Y) + c_2(Y) + b_1(Y) - c_2(X) - b_1(X) = a_1(X) - b_1(X) + c_2(Y) - a_2(Y) < 0,$$

which implies that A is not implementable with verification. □

Definition 48 (strongly 2D-stable) *A social function A is* strongly 2D-stable *if, for all i, for all $\mathbf{b}_{-i}$, and for all unnatural edges $(\mathbf{a},\mathbf{b})$, it holds that $a_1(A_{\mathbf{b}_{-i}}(\mathbf{b})) = b_1(A_{\mathbf{b}_{-i}}(\mathbf{b}))$ or $a_2(A_{\mathbf{b}_{-i}}(\mathbf{b})) = b_2(A_{\mathbf{b}_{-i}}(\mathbf{b}))$ implies $A_{\mathbf{b}_{-i}}(\mathbf{a}) = A_{\mathbf{b}_{-i}}(\mathbf{b})$.*

It turns out that the above condition is a sufficient one as stated by the following theorem (which proof will be given in the full version of the paper).

Theorem 49 *Every strongly 2D-stable social choice function A, dealing with bi-dimensional valuations, is implementable with verification.*