Two-Way Greedy: Algorithms for Imperfect Rationality

Diodato Ferraioli[1], Paolo Penna[2], and Carmine Ventre[3](✉)

[1] Università di Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano, SA, Italy
dferraioli@unisa.it
[2] ETH Zürich, Rämistrasse 101, 8092 Zürich, Switzerland
paolo.penna@inf.ethz.ch
[3] King's College London, Strand, London WC2R 2LS, UK
carmine.ventre@kcl.ac.uk

Abstract. The realization that selfish interests need to be accounted for in the design of algorithms has produced many interesting and valuable contributions in computer science under the general umbrella of algorithmic mechanism design. Novel algorithmic properties and paradigms have been identified and studied in the literature. Our work stems from the observation that selfishness is different from rationality; agents will attempt to strategize whenever they perceive it to be convenient according to their imperfect rationality. Recent work in economics [18] has focused on a particular notion of imperfect rationality, namely absence of contingent reasoning skills, and defined obvious strategyproofness (OSP) as a way to deal with the selfishness of these agents. Essentially, this definition states that to care for the incentives of these agents, we need not only pay attention about the relationship between input and output, but also about the way the algorithm is run. However, it is not clear to date what algorithmic approaches ought to be used for OSP. In this paper, we rather surprisingly show that, for binary allocation problems, OSP is fully captured by a natural combination of two well-known and extensively studied algorithmic techniques: forward and reverse greedy. We call two-way greedy this underdeveloped algorithmic design paradigm.

Our main technical contribution establishes the connection between OSP and two-way greedy. We build upon the recently introduced cycle monotonicity technique for OSP [9]. By means of novel structural properties of cycles and queries of OSP mechanisms, we fully characterize these mechanisms in terms of extremal implementations. These are protocols that ask each agent to consistently separate one extreme of their domain at the current history from the rest. Through the natural connection with the greedy paradigm, we are able to import a host of known approximation bounds to OSP and strengthen the strategic properties of this family of algorithms. Finally, we begin exploring the full power of two-way greedy (and, in turns, OSP) in the context of set systems.

Diodato Ferraioli is supported by GNCS-INdAM and the Italian MIUR PRIN 2017 Project ALGADIMAR "Algorithms, Games, and Digital Markets". Carmine Ventre acknowledges funding from the UKRI Trustworthy Autonomous Systems Hub (EP/V00784X/1).

M. Feldman et al. (Eds.): WINE 2021, LNCS 13112, pp. 3–21, 2022.
https://doi.org/10.1007/978-3-030-94676-0_1

1 Introduction

An established line of work in computer science recognizes the important role played by self interests. If ignored, these self interests can misguide the algorithm or protocol at hand and lead to suboptimal outcomes. Mechanism design has emerged as the framework of reference to deal with this selfishness. Mechanisms are protocols that interact with the selfish agents involved in the computation; the information elicited through this interaction is used to choose a certain outcome (via an algorithm). The goal of a mechanism is that of reconciling the potentially contradictory aims of agents with that of the designer (i.e., optimize a certain objective function). The agents attach a utility (typically defined as quasi-linear function of the *transfers* defined by the mechanism and the agent's *type* – i.e., cost or valuation – for the solution) to each outcome and are therefore incentivized to force the output of an outcome that maximizes their utility (rather than maximizing the objective function). The quality of a mechanism is assessed against how well it can approximate the objective function whilst giving the right incentives to the agents.

In this context, one seeks to design *strategyproof (SP) mechanisms*—these guarantee that agents will not strategize as it will be in their best interest to adhere to the rules set by the mechanism—and aims to understand what is the best possible approximation that can be computed for the setting of interest. For example, it is known how for *utilitarian* problems (roughly speaking, those whose objective function is the sum of all the agents' types) it is possible to simultaneously achieve optimality and strategyproofness, whilst some non-utilitarian objective (such as, min-max) cannot be approximated too well (irrespectively of computational considerations), see, e.g., [21]. These results can be proved purely from an algorithmic perspective – that ignores incentives and selfishness – in that it is known how strategyproofness is equivalent to a certain *monotonicity property* of the algorithm used by the mechanism to compute the outcome. This monotonicity relates the outcomes of two instances, connected by SP constraints, and limits what the algorithm can do on them. For example, if an agent is part of the solution computed on instance I and becomes "better" (e.g., faster) in instance I' then the algorithm must select the agent also in the solution returned for instance I', all other things unchanged.

Recent research in mechanism design has highlighted how cognitive limitations of the agents might mean that SP is too weak a desideratum for mechanisms. Even for the simplest setting of one-item auction, there is experimental evidence that people strategize against the sealed-bid implementation of second-price auction, whilst ascending-price auction seems easier to grasp [1,14]. The concept of *obvious strategyproofness* (OSP) has been defined in [18] to capture this particular form of imperfect rationality, which is shown to be equivalent to the absence of contingent reasoning skills. Intuitively, for an agent it is obvious to understand if a strategy is better than another in that the worst possible outcome for the former is better than the best one for the latter.

Can we, similarly to SP, derive bounds on the quality of OSP mechanisms that are oblivious to strategic considerations?

There are two obstacles to getting a fully algorithmic approach to OSP mechanisms due to their structure. Whereas SP mechanisms are pairs comprised of an algorithm and transfer (a.k.a., payment) function, in OSP we have a third component (the so-called *implementation tree*) which encapsulates the execution details (e.g., sealed bid vs ascending price) of the mechanisms and the obviousness of the strategic constraints. (For OSP, in fact, the implementation details matter and the classical Revelation Principle does not hold [18].) A technique, known as cycle monotonicity (CMON), allows to express the existence of SP payments for an algorithm in terms of the weight of the cycles in a suitably defined graph. Specifically, it is known that it is sufficient to look at cycles of length two for practically all optimization problems of interest [22]—this yields the aforementioned property of monotone algorithms. Recent work [8,9] extends CMON to OSP and allows to focus only on algorithms and implementation trees. Whilst this has allowed some progress towards settling our main question in the context of single-parameter agents, some unsatisfactory limitations are still present. Firstly, handling two interconnected objects, namely algorithm and implementation tree, simultaneously is hard to work with: e.g., novel ad-hoc techniques (dubbed CMON two-ways in [9]) had to be developed to prove lower bounds. Secondly, the CMON extension to OSP is shown to require the study of cycles of any length, thus implying that the "monotonicity" of the combination algorithm/implementation tree needs to hold amongst an arbitrary number of instances, as opposed to two as in the case of SP. Thirdly, the mechanisms constructed in [8,9] only work for three-value domains since they rely on the simpler two-instance monotonicity (referred to as monotonicity henceforth).

Our Contributions. The technical challenge left open by previous work was to relate monotonicity to many-instance monotonicity. In this paper, we solve this challenge by providing a characterization of OSP mechanisms for binary allocation problems (for which the outcome for each agent is either to be selected or not). This enables us to show that the shape of the implementation tree is essentially fixed and answer the question above in the positive. It turns out that the exact algorithmic structure of OSP mechanisms is intimately linked with a (slight generalization of a) well known textbook paradigm:

OSP can be achieved if and only if the algorithm is two-way greedy.

What does it mean for an algorithm to be two-way greedy? The literature in computer science and approximation algorithms has extensively explored what we call *forward greedy*. These are algorithms that use a (possibly adaptive) (in-)priority function and incrementally build up a solution by adding therein the agent with the highest priority, if this preserves feasibility. It is known that if the priority rule is monotone in each agent's type then this leads to a SP direct-revelation mechanism (see, e.g., [17]). What we show here is that the strategyproofness guarantee is actually much stronger and can deal with imperfect

rationality. This is achieved with a simple implementation of forward greedy that sweeps through each agent's domain from the best possible type to the worst. Another relevant approach known in the literature is Deferred Acceptance auctions (DAAs) or reverse greedy algorithms [7,12]. These use a (possibly adaptive) (out-)priority function and build a feasible solution by incrementally throwing out the agents whose type is not good enough with respect the current priority (i.e., whose cost (valuation) is higher (lower) than the out-priority) until a feasible solution is found. It is already known that DAAs are OSP [20] but not the extent to which focusing on them would be detrimental to finding out the real limitations of OSP mechanisms. Two-way greedy algorithms *combine in- and out-priorities*; each agent faces either a greedy in- or out-priority; in the former case, they are included in the solution if feasibility is preserved while in the latter they are excluded from it if the current solution is not yet feasible. The direction faced can depend on which agents have been included in or thrown out from the eventual solution at that point of the execution; in this sense, these are particular adaptive priority algorithms. For a formal definition, please see Sect. 4 and Algorithm 3.

Two-way greedy algorithms stem from our characterization of OSP mechanisms in terms of "extremal implementation trees"; roughly speaking, in these mechanisms we always query each agents about *(the same) extreme* of their domain at the current history. To prove this characterization, we first give a couple of structural properties of OSP mechanisms. We specifically show (i) when a query can be made to guarantee OSP; and, (ii) how a mechanism that is monotone but not many-instance monotone looks like. We use the former property to show that, given an OSP mechanism, we can modify the structure of its implementation tree to make it extreme whilst guaranteeing that the many-instance monotonicity is preserved (i.e., the structure (ii) is not possible). We also show that extremal mechanisms are monotone and that structure (ii) can never arise, thus proving the sufficient condition of our characterization. One caveat about these extremal mechanisms and two-way greedy is necessary. This has to do with a technical exception to the rule of never interleaving top queries (asking for the maximum of the current domain) with bottom queries (asking for the minimum) to an agent. An OSP mechanism can in fact interleave those when, at the current history, an agent becomes *revealable*, that is, the threshold separating winning bids from losing ones, becomes known. In other words, this is a point in which the outcome for this agent (but not necessarily the entire solution) is determined for all but one of her types. OSP mechanisms can at this point use any query ordering to find out what the type of the agent is; this does not affect the incentives of the agents. Accordingly, in a two-way greedy algorithm an agent can face changes of priority direction (e.g., from in- to out-priority) in these circumstances.

To the best of our knowledge this is one of the first known cases of a relationship between strategic properties and an algorithmic paradigm, as opposed to a property about the solution output by the algorithm. Two possible interpretations of this connection can be given. On a conceptual level, the fact that the

Revelation Principle does not hold true for OSP means that we care about the implementation details and thus the right algorithmic nature has to be paradigmatic rather than being only about the final output. On a technical level, given that monotonicity (i.e., two-cycles) is not sufficient for OSP, the need to study many-instance monotonicity (i.e., any cycle) requires to go beyond an output property and look for the way in which the algorithm computes *any* solution.

Our OSP characterization is related to the Personal Clock Auctions (PCAs) in [18]. Roughly speaking, Li proves that for binary allocation problems, each agent faces either an ascending-price auction (where there is an increasing transfer going rate to be included in the solution) or a descending-price auction (where there is a decreasing transfer going rate to be excluded from the solution). There are conceptual and technical differences between our characterization and Li's. His focus is on characterizing the auction format (i.e., social choice function, payments and implementation tree) whereas ours concentrates on algorithms. Studying the approximation guarantee of PCAs requires to disentangle these three components. We defer to the full version of this paper a concrete example taken from the corrigendum of [18] that shows how the technical definition of PCA (contrarily to its intuitive and informal description) does not allow for a simple algorithmic characterization in terms of greedy. It turns out that PCAs require to reason about strategies over the extensive-form implementation as opposed to type profiles – this makes them unsuitable and underspecified from the algorithmic point of view. From the technical perspective, Li requires *continuous domains* whereas we assume that these are *finite*, mainly because of the inherent limitations of the OSP CMON technique [9]. For one, our setup is arguably more interesting for OSP, as it is notoriously harder to understand how to execute extensive-form games in the continuous case. Secondly, our proof technique cannot rely on the existence of a unique threshold (there are two threshold values in discrete domains, i.e., extreme winning and losing reports do not "meet" in the limit) unlike [18]. Importantly, our results allow for a more workable notion of OSP mechanisms for binary allocation problems; our approach and terminology are closer to computer science and algorithms and give a specific recipe to reason about design and analysis of these mechanisms.

We give a host of bounds on the approximation guarantee of OSP mechanisms by relying on our characterization and the known approximation guarantees of forward greedy algorithms, cf. Table 1 below. The strategic equivalence of forward and reverse greedy is one of the most far reaching consequences of our results, given (i) the rich literature on the approximation of forward greedy, and (ii) the misconception about the apparent weaknesses of accepting, rather than rejecting, auctions [20] (see Sect. 4). We expect our work to spawn further research about OSP, having fully extracted the algorithmic nature of these mechanisms. The power and limitations of OSP can now be fully explored, in the context of binary allocation problems. We present some initial bounds on the quality of these algorithms/mechanisms (see Sect. 4). Notably, we close the gap for the approximation guarantee of OSP mechanisms for the knapsack auctions studied in [7]. We show that the logarithmic upper bound provided therein is

basically tight, not just for reverse greedy (as shown in [7]) but for the whole class.

Since our main objective is that of establishing the power of OSP mechanisms, in terms of algorithmic tools and their approximation, we do not primarily focus on their computational complexity. Consequently, our lower bounds are unconditional. We discuss this aspect and more opportunities for further research in the conclusions (see Sect. 5). The proofs missing due to lack of space are deferred to the full version of this paper.

2 Preliminaries and Notation

We define a set N of n *selfish agents* and a set of feasible *outcomes* $\mathcal{S}$. Each agent i has a *type* $t_i \in D_i$, where D_i is the *domain* of i. The type t_i is assumed to be *private knowledge* of agent i. We let $t_i(X) \in \mathbb{R}$ denote the *cost* of agent i with type t_i for the outcome $X \in \mathcal{S}$. When costs are negative, it means that the agent has a profit from the solution, called *valuation*.

A *mechanism* has to select an outcome $X \in \mathcal{S}$. For this reason, the mechanism interacts with agents. Specifically, agent i takes *actions* (e.g., saying yes/no) that may depend on her presumed type $b_i \in D_i$ (e.g., saying yes could "signal" that the presumed type has some properties that b_i enjoys). To stress this we say that agent i takes *actions compatible with (or according to)* b_i Note that the presumed type b_i can be different from the real type t_i. For a mechanism M, we let $M(\mathbf{b})$ denote the outcome returned by M when agents take actions according to their presumed types $\mathbf{b} = (b_1, \ldots, b_n)$ (i.e., each agent i takes actions compatible with the corresponding b_i). This outcome is given by a pair (f, p), where $f = f(\mathbf{b}) = (f_1(\mathbf{b}), \ldots, f_n(\mathbf{b}))$ (termed *social choice function* or, simply, algorithm) maps the actions taken by the agents according to $\mathbf{b}$ to a feasible solution in $\mathcal{S}$, and $p = p(\mathbf{b}) = (p_1(\mathbf{b}), \ldots, p_n(\mathbf{b})) \in \mathbb{R}^n$ maps the actions taken by the agents according to $\mathbf{b}$ to *payments*. Note that payments need not be positive.

Each selfish agent i is equipped with a *quasi-linear utility function* $u_i \colon D_i \times \mathcal{S} \to \mathbb{R}$: for $t_i \in D_i$ and for an outcome $X \in \mathcal{S}$ returned by a mechanism M, $u_i(t_i, X)$ is the utility that agent i has for the implementation of outcome X when her type is t_i, i.e., $u_i(t_i, M(b_i, \mathbf{b}_{-i})) = p_i(b_i, \mathbf{b}_{-i}) - t_i(f(b_i, \mathbf{b}_{-i}))$. In this work we will focus on *single-parameter* settings, that is, the case in which the private information of each bidder i is a single real number t_i and $t_i(X)$ can be expressed as $t_i w_i(X)$ for some publicly known function w_i. To simplify the notation, we will write $t_i f_i(\mathbf{b})$ when we want to express the cost of a single-parameter agent i of type t_i for the output of social choice function f on input the actions corresponding to a bid vector $\mathbf{b}$. In particular, we will consider *binary allocation problems*, where $f_i(\mathbf{b}) \in \{0, 1\}$, i.e., each agent either belongs to the returned solution ($f_i(\mathbf{b}) = 1$) or not ($f_i(\mathbf{b}) = 0$). A class of binary allocation problems of interest are *set systems* $(E, \mathcal{F})$, where E is a set of elements and $\mathcal{F} \subseteq 2^E$ is a family of feasible subsets of E. Each element $i \in E$ is controlled by a selfish agent, that is, the cost for including i is known only to agent i and is

equal to some value t_i. The social choice function f must choose a feasible subset of elements X in $\mathcal{F}$ that minimizes $\sum_{i=1}^{n} b_i(X)$. When the t_i's are non-negative (non-positive, respectively) then this objective is called social cost minimization (social welfare maximization, respectively).

Extensive-Form Mechanisms and Obvious Strategyproofness. We now introduce the concept of implementation tree and we formally define (deterministic) obviously strategy-proof mechanisms. Our definition is built on [19] rather than the original definition in [18]. Specifically, our notion of implementation tree is equivalent to the concept of round-table mechanisms in [19], and our definition of OSP is equivalent to the concept of SP-implementation through a round table mechanism, that is proved to be equivalent to the original definition.

Let us first formally model how a mechanism works. An *extensive-form mechanism* M is a triple $(f, p, \mathcal{T})$ where, as above, the pair (f, p) determines the outcome of the mechanism, and $\mathcal{T}$ is a tree, called *implementation tree*, s.t.:

- Every leaf ℓ of the tree is labeled with a possible outcome of the mechanism $(X(\ell), p(\ell))$, where $X(\ell) \in \mathcal{S}$ and $p(\ell) \in \mathbb{R}$;
- Each node u in the implementation tree $\mathcal{T}$ defines the following:
  - An agent $i = i(u)$ to whom the mechanism makes some query. Each possible answer to this query leads to a different child of u.
  - A subdomain $D^{(u)} = (D_i^{(u)}, D_{-i}^{(u)})$ containing all types that are *compatible* with u, i.e., with all the answers to the queries from the root down to node u. Specifically, the query at node u defines a partition of the current domain of i, $D_i^{(u)}$ into $k \geq 2$ subdomains, one for each of the k children of node u. Thus, the domain of each of these children will have as the domain of i, the subdomain of $D_i^{(u)}$ corresponding to a different answer of i at u, and an unchanged domain for the other agents.

Observe that, according to the definition above, for every profile $\mathbf{b}$ there is only one leaf $\ell = \ell(\mathbf{b})$ such that $\mathbf{b}$ belongs to $D^{(\ell)}$. Similarly, to each leaf ℓ there is at least a profile $\mathbf{b}$ that belongs to $D^{(\ell)}$. For this reason, we say that $M(\mathbf{b}) = (X(\ell), p(\ell))$. Two profiles $\mathbf{b}$, $\mathbf{b}'$ are said to *diverge* at a node u of $\mathcal{T}$ if this node has two children v, v' such that $\mathbf{b} \in D^{(v)}$, whereas $\mathbf{b}' \in D^{(v')}$. For every such node u, we say that $i(u)$ is the *divergent agent* at u.

Definition 1 (OSP mechanisms). *An extensive-form mechanism M is* obviously strategy-proof (OSP) *if for every agent i with real type t_i, for every vertex u such that $i = i(u)$, for every $\mathbf{b}_{-i}, \mathbf{b}'_{-i}$ (with $\mathbf{b}'_{-i}$ not necessarily different from $\mathbf{b}_{-i}$), and for every $b_i \in D_i$, with $b_i \neq t_i$, s.t. $(t_i, \mathbf{b}_{-i})$ and $(b_i, \mathbf{b}'_{-i})$ are compatible with u, but diverge at u, it holds that $u_i(t_i, M(t_i, \mathbf{b}_{-i})) \geq u_i(t_i, M(b_i, \mathbf{b}'_{-i}))$.*

Roughly speaking, OSP requires that, at each time step agent i is asked to take a decision that depends on her type, the worst utility that she can get if she behaves according to her true type is at least the best utility she can get by behaving differently. We stress that our definition does not restrict the alternative behavior to be consistent with a fixed type. Each leaf of the tree

rooted in u, denoted $\mathcal{T}_u$, corresponds to a profile $\mathbf{b} = (b_i, \mathbf{b}'_{-i})$ compatible with u: then, our definition implies that the utility of i in the leaves where she plays truthfully is at least as much as the utility in every other leaf of $\mathcal{T}_u$.

Cycle-Monotonicity Characterizes OSP Mechanisms. We next describe the main tools in [9] showing that OSP can be characterized by the absence of negative-weight cycles in a suitable weighted graph over the possible strategy profiles. For ease of exposition, we will focus on non-negative costs but the results hold no matter the sign. We consider a mechanism M with implementation tree $\mathcal{T}$ for a social choice function f, and define the following concepts:

- **Separating Node:** A node u in the implementation tree $\mathcal{T}$ is $(\mathbf{a}, \mathbf{b})$-separating for agent $i = i(u)$ if $\mathbf{a}$ and $\mathbf{b}$ are compatible with u (that is, $\mathbf{a}, \mathbf{b} \in D^{(u)}$), and the two types a_i and b_i belong to two different subdomains of the children of u (thus implying $a_i \neq b_i$).
- **OSP-graph:** For every agent i, we define a directed weighted graph $\mathcal{O}_i^{\mathcal{T}}$ having a node for each profile in $D = \times_i D_i$. The graph contains edge $(\mathbf{a}, \mathbf{b})$ if and only if $\mathcal{T}$ has some node u which is $(\mathbf{a}, \mathbf{b})$-separating for $i = i(u)$, and the weight of this edge is $w(\mathbf{a}, \mathbf{b}) = a_i(f_i(\mathbf{b}) - f_i(\mathbf{a}))$. Throughout the paper, we will denote with $\mathbf{a} \to \mathbf{b}$ an edge $(\mathbf{a}, \mathbf{b}) \in \mathcal{O}_i^{\mathcal{T}}$, and with $\mathbf{a} \rightsquigarrow \mathbf{b}$ a path among these two profiles in $\mathcal{O}_i^{\mathcal{T}}$.
- **OSP Cycle Monotonicity (OSP CMON):** We say that the OSP cycle monotonicity (OSP CMON) holds if, for all i, the graph $\mathcal{O}_i^{\mathcal{T}}$ does not contain negative-weight cycles. Moreover, we say that the OSP two-cycle monotonicity (OSP 2CMON) holds if the same is true when considering cycles of length two only, i.e., cycles with only two edges.

Theorem 1 ([9]). *A mechanism with implementation tree $\mathcal{T}$ for a social function f is OSP on finite domains if and only if OSP CMON holds.*

3 A Characterization of OSP Mechanisms

Given the theorem above, we henceforth assume that the agents have finite domains. In this section we present our characterization of OSP mechanisms for binary allocation problems. We begin by setting the scene and then give three useful structural properties of OSP mechanisms.

Observation 1 (Basic Properties of the OSP-graph). *The weight of an edge $(\mathbf{a}, \mathbf{b})$ is non-zero for: $f_i(\mathbf{a}) = 0$ and $f_i(\mathbf{b}) = 1$, in which case the weight is $w(\mathbf{a}, \mathbf{b}) = +a_i$ (**positive-weight edge**); or, $f_i(\mathbf{a}) = 1$ and $f_i(\mathbf{b}) = 0$, in which case the weight is $w(\mathbf{a}, \mathbf{b}) = -a_i$ (**negative-weight edge**). If OSP 2CMON holds, then (i) for every positive-weight edge as above, we have $b_i < a_i$ and, by symmetry, (ii) for every negative-weight edge as above, we have $a_i < b_i$. Note that these inequalities are strict since $a_i \neq b_i$ for every edge as above.*

Definition 2 (0-always, 1-always, unclear). *For* sel $\in \{0, 1\}$, *indicating whether i is selected, a generic type $t_i \in D_i^{(u)}$, and a node u of the implementation tree, we define: (i) t_i is* sel*-always if $f_i(\mathbf{t}) =$* sel *for all $\mathbf{t}_{-i} \in D_{-i}^{(u)}$; (ii) t_i is*

sel-*sometime if* $f_i(\mathbf{t}) = \text{sel}$ *for some* $\mathbf{t}_{-i} \in D_{-i}^{(u)}$. *Moreover,* t_i *is unclear if it is neither* 0-*always nor* 1-*always, i.e., both* 0-*sometime and* 1-*sometime.*

Structure of the Implementation Tree. We first observe that, w.l.o.g., we can always assume that each node u in the implementation tree has two children.

Observation 2. *For any OSP mechanism* $M = (f, p, \mathcal{T})$ *where* $\mathcal{T}$ *is not a binary tree, there is an OSP mechanism* $M' = (f, p, \mathcal{T}')$ *where* $\mathcal{T}'$ *is a binary tree.*

The mechanism then partitions $D_i^{(u)}$ into two subdomains $L^{(u)}$ and $R^{(u)}$. In general, the two parts $L^{(u)}$ and $R^{(u)}$ can be *any* partition of the subdomain $D_i^{(u)}$, and they are not necessarily ordered. For example, if $D_i^{(u)} = \{1, 2, 5, 6, 7\}$, a query "is your type even?" results in $L^{(u)} = \{1, 5, 7\}$ and $R^{(u)} = \{2, 6\}$, with these two subdomains being "incomparable". However, we will see that in an OSP mechanism these sets $L^{(u)}$ and $R^{(u)}$ share a special structure.

Structure of Admissible Queries. We begin with a simple observation about the structure of the parts implied by a simple application of OSP 2CMON.

Observation 3. *Assume OSP 2CMON holds. At every node* u *where the subdomain is separated into two parts* L *and* R *the following holds. Every* 0-*sometime type* l *in one side (say* L*) implies that all types in the other side that are bigger* ($r \in R$ *with* $r > l$) *must be* 0-*always. Similarly, every* 1-*sometime type* l *in one side implies that all types in the other side that are smaller must be* 1-*always.*

Next lemma characterizes the admissible queries in OSP mechanisms.

Lemma 1 (Admissible Queries). *Let* M *be an OSP mechanism with implementation tree* $\mathcal{T}$ *and let* u *be a node of* $\mathcal{T}$ *where the query separates the current subdomain into* L *and* R*. Then one of the following conditions must hold:*

1. *At least one of the two parts, some* $P \in \{L, R\}$*, is homogeneous meaning that either all* $p \in P$ *are* 0-*always or all* $p \in P$ *are* 1-*always.*
2. *Agent* i *is revealable at node* u*, meaning that* $D_i^{(u)}$ *has the following structure:*

$$D_i^{(u)} = \{\underbrace{d_1 < d_2 < \cdots < d_a}_{1\text{-}always} < d_* < \underbrace{d'_1 < d'_2 < \cdots d'_b}_{0\text{-}always}\} \tag{1}$$

where each subset of 1-*always and* 0-*always types may be empty, and* d_* *may or may not be* 1-*always or* 0-*always. Moreover, the two parts* $P \in \{L, R\}$ *must have the following structure:*

$$P = \{\underbrace{p_1 < \cdots < p_a}_{1\text{-}always} < p_* < \underbrace{p'_1 < \cdots < p'_b}_{0\text{-}always}\} \tag{2}$$

with at most one type in $D_i^{(u)}$ *being unclear (neither* 0-*always nor* 1-*always).*

Structure of Negative-Weight Cycles. A crucial step is to provide a simple *local-to-global* characterization of OSP mechanisms which (essentially) involves only four type profiles. The following theorem states that, if there is a negative-weight cycle with *more than two* edges but no length-two negative cycle (OSP CMON is violated but OSP 2CMON holds), then there exists a cycle with a rather *special structure*, and this special structure is fully specified by only *four* profiles (the cycle itself may involve several profiles though).

Theorem 2 (Four-Profile Characterization). *Let M be a mechanism with implementation tree $\mathcal{T}$ and social choice function f that is OSP 2CMON but not OSP CMON. Then, every negative-weight cycle C in some OSP graph $\mathcal{O}_i^{\mathcal{T}}$ is of the following form: $C = \mathbf{b}^{(2)} \to \mathbf{b}^{(1)} \rightsquigarrow \mathbf{b}^{(3)} \to \mathbf{b}^{(4)} \rightsquigarrow \mathbf{b}^{(2)}$ where these four profiles satisfy (i) $b_i^{(1)} < b_i^{(2)} < b_i^{(3)} < b_i^{(4)}$, (ii) $f_i(\mathbf{b}^{(1)}) = f_i(\mathbf{b}^{(3)}) = 1$, and (iii) $f_i(\mathbf{b}^{(2)}) = f_i(\mathbf{b}^{(4)}) = 0$. Moreover, there is no edge between $\mathbf{b}^{(2)}$ and $\mathbf{b}^{(3)}$ in $\mathcal{O}_i^{\mathcal{T}}$.*

This characterization will enable us to provide a simple local transformation of the queries of an OSP mechanisms where we use only top or bottom queries.

Definition 3 (Top and Bottom Queries). *Let $i = i(u)$ for a node u of the implementation tree. If the query at u partitions $D_i^{(u)}$ into $\{\min D_i^{(u)}\}$ and $D_i^{(u)} \setminus \{\min D_i^{(u)}\}$ then we call the query at u a bottom query. A top query at u, instead, separates the maximum of $D_i^{(u)}$ from the rest.*

3.1 OSP is Equivalent to Weak Interleaving

In this section, we show that without loss of generality, we can focus on OSP mechanisms where each agent is asked only top queries or only bottom queries, except when her type becomes revealable (Condition 2. in Lemma 1). In that sense, these mechanisms interleave top and bottom queries for an agent only in a "weak" form.[1] Specifically, let us begin by providing the following definition.

Definition 4 (Extremal, No Interleaving, Weak Interleaving). *A mechanism is extremal if every query is a bottom query or a top query (both types of queries may be used for the same agent).*

An extremal mechanism makes no interleaving queries if each agent is consistently asked only top queries or only bottom queries at each history where she is divergent (some agents may be asked top queries only, and other agents bottom queries only).

A weak interleaving mechanism satisfies the condition that, if top queries and bottom queries are interleaved for some agent i at some node u in the implementation tree, then agent i is revealable at u in the sense of Condition 2. in Lemma 1.

[1] It may appear that an alternative formalization of the interleaving between in- and out-priorities could be a query where the type is fully revealed; this would not work as there is still one type for which the outcome is undetermined.

Theorem 3. *For each binary outcome problem, an OSP mechanism exists if and only if an extremal mechanism with weak interleaving exists.*

The Proof. We start with the necessary condition and prove the following chain of implications: *OSP mechanism* $\Rightarrow$ *OSP extremal mechanism* $\Rightarrow$ *OSP weak interleaving mechanism*. Intuitively, given the structure of admissible queries in Lemma 1, we show below that we can locally replace every query with a homogeneous part (Condition 1.) by a "homogeneous" sequence of only top queries or only bottom queries. Moreover, these queries can also be used when the agent becomes revealable (Condition 2.). To this aim, we use the four-profile characterization of negative-weight cycles in Theorem 2.

Theorem 4. *Any OSP mechanism* $M = (f, p, \mathcal{T})$ *can be transformed into an equivalent extremal OSP mechanism* $M' = (f, p, \mathcal{T}')$.

Proof Sketch. Since the mechanism M is OSP, then OSP CMON must hold (Theorem 1). Let u be a node of $\mathcal{T}$ where M is not extreme, and let $i = i(u)$ be the corresponding divergent agent at u. Assume that all previous queries of i in the path from the root to u are extremal (if not, we can apply the argument to the first query that is not extremal and reiterate).

We locally modify $\mathcal{T}$ in order to make a suitable sequence of bottom queries and top queries about a certain subset Q of types, before we make any further query in the two subtrees of u. It can be shown that this local modification does not affect OSP CMON of agents different from i and it preserves OSP 2CMON for all agents. The proof uses the following main steps and key observations:

1. If OSP CMON is no longer true for $\mathcal{T}'$, then there exists a negative-weight cycle C' which was not present in $\mathcal{O}_i^{\mathcal{T}}$ and therefore must use some added edges $\mathbf{a}^{(1)} \to \mathbf{a}^{(2)}$ that were not present in $\mathcal{O}_i^{\mathcal{T}}$ and that have been added to $\mathcal{O}_i^{\mathcal{T}'}$ because of the new queries for types in Q;
2. The negative-weight cycle C' must be of the form specified by Theorem 2; in particular, the edge $\mathbf{b}^{(2)} \to \mathbf{b}^{(3)}$ does not exist in $\mathcal{O}_i^{\mathcal{T}'}$ (which helps to determine properties of the four profiles characterizing C');
3. We use the following *bypass argument* to conclude that in the original graph there is a negative-weight cycle, thus contradicting OSP CMON of the original mechanism. Specifically, for every added edge $\mathbf{a}^{(1)} \to \mathbf{a}^{(2)}$, the original graph $\mathcal{O}_i^{\mathcal{T}}$ contains a bypass path $\mathbf{a}^{(1)} \to \mathbf{b}^{(bp)} \to \mathbf{a}^{(2)}$ such that $w(\mathbf{a}^{(1)} \to \mathbf{b}^{(bp)} \to \mathbf{a}^{(2)}) \leq w(\mathbf{a}^{(1)} \to \mathbf{a}^{(2)})$. By replacing every added edge in C' with the corresponding bypass path, we get a cycle C with negative weight, $w(C) \leq w(C') < 0$. □

We are now ready to show that one can think of weak interleaving OSP mechanisms without loss of generality.

Theorem 5. *Let M be an OSP extremal mechanism with implementation tree $\mathcal{T}$. For any node $u \in \mathcal{T}$, if agent $i = i(u)$ is not revealable at node u, then M has no interleaving for agent i at u.*

The proof of Theorem 3 is completed below with the sufficiency.

Theorem 6. *An extremal mechanism with weak interleaving is OSP.*

4 Two-Way Greedy Algorithms

We reiterate that our characterization does not require costs to be positive; so it holds true for negative costs, that is, valuations. We will then now talk simply about type when we refer to the agents' private information, be it costs or valuations. We will also sometimes use the terminology of in-query (out-query, respectively) to denote a bottom (top) query for costs and a top (bottom) query for valuations.

Definition 5 ((Anti-)Monotone Functions). *We say that a function is monotone (antimonotone, resp.) in the type if it is decreasing (increasing, resp.) in the cost, and increasing (decreasing, resp.) in the valuation.*

In this section, we translate our characterization into algorithmic insights on OSP mechanisms and show a connection between their format and a certain family of adaptive priority algorithms that they use. As a by-product, we show the existence of a host of new mechanisms. We are able to provide the first set of upper bounds on the approximation guarantee of OSP mechanisms, that are independent from domain size (as in [8,9]) or assumptions on the designer's power to catch and punish lies (as in [10,11]), see Table 1. The table also contains the new bounds we can prove on the approximation of OSP mechanisms, by leveraging our algorithmic characterization (i.e., Theorems 7–10).

Table 1. Bounds on the approximation guarantee of OSP mechanisms. (The result for CAs has been observed in [6]. The mechanisms for matroids also follows from Corollary 2.)

Problem	Bound
Known Single-Minded Combinatorial Auctions (CAs)	$\sqrt{m}$ ([17] + Corollary 1)
MST (& weighted matroids)	1 ([15] + Corollary 1)
Max Weighted Matching	2 ([2] + Corollary 1)
p-systems[‡]	p ([13] + Corollary 1)
Weighted Vertex Cover	2 ([5] + Corollary 1)
Shortest Path	∞ (Theorem 7)
Restricted Knapsack Auctions	$\Omega(\sqrt{n})$ (Theorem 8)
Asymmetric Restricted Knapsack Auctions (3 values)	$\sqrt{n-1}$ (Theorem 9)
Knapsack Auctions	$\Omega(\sqrt{\ln n})$ (Theorem 10)

[‡]A p-system is a downward-closed set system $(E, \mathcal{F})$ where there are at most p *circuits*, that is, minimal subsets of E not belonging to $\mathcal{F}$ [13].

Immediate-Acceptance Auctions and Forward Greedy. Let us begin by discussing forward greedy.

Definition 6 (Forward Greedy). *A forward greedy algorithm uses functions* $g_i^{(in)} : \mathbb{R} \times \mathbb{R}^k \to \mathbb{R}$, $k \leq n-1$, *to rank the bids of the players and builds a solution by iteratively adding the agent with highest rank if that preserves feasibility (cf. Algorithm 1). A forward greedy algorithm is monotone if each* $g_i^{(in)}$ *is monotone in* i*'s private type (i.e., its first argument).*

Algorithm 1: Forward greedy algorithm

```
1  Let b_1, ..., b_n the input bids
2  P ← F (P is the set of all feasible solutions)
3  I ← ∅ (I is the set of infeasible agents, i.e., agents that cannot be included in
   the final solution)
4  A ← N (A is the set of active or infeasible agents)
5  while |P| > 1 do
6      Let i = arg max_{k∈A\I} g_k^{(in)}(b_k, b_{N\A})
7      if there are solutions S in P such that i ∈ S then
8          Drop from P all the solutions S such that i ∉ S (if any)
9          A ← A \ {i}
10     else
11         I ← I ∪ {i}
12 Return the only solution in P
```

A few observations are in order for Algorithm 1. Firstly, it is not too hard to see that it will always return a solution (i.e., there will eventually be a unique feasible solution in $\mathcal{P}$). To see this consider the case in which (at least) two solutions S, S' are in $\mathcal{P}$. Then there must exist an agent j such that $j \in S \Delta S'$, Δ denoting the symmetric difference between sets. This means that $j \in \mathcal{A} \setminus \mathcal{I}$ and the forward greedy algorithm will decide in the next steps whether j is part of the solution or not (and consequently whether to keep S or S'). Secondly, we stress how forward greedy algorithms belong to the family of adaptive priority algorithms [4]. At Line 6, Algorithm 1 (potentially) updates the priority as a function of the bids of those bidders who have left the auction (i.e., that are not active anymore). A peculiarity of the algorithm (to do with its OSP implementation) is that the adaptivity does not depend on bidders who despite their high priority cannot be part of the eventual solution (i.e., the agents we add to $\mathcal{I}$).[2] This distinction would not be necessary if the problem at hand were *upward closed* (i.e., if a solution S is feasible then any $S' \supset S$ would be in $\mathcal{F}$ too). Thirdly, a notable subclass of forward greedy algorithms are fixed-priority algorithms, where Line 6 and the while loop are swapped (and priority functions only depend on the agents' types). (These algorithms do not need to keep record

[2] Note that a syntactically (but not semantically) alternative definition of forward greedy algorithms could do without $\mathcal{I}$ by requiring an extra property on the priority functions (i.e., adaptively floor all the priorities of infeasible players).

of $\mathcal{I}$ either.) This algorithmic paradigm has been applied to different optimization problems (e.g., Kruskal's Minimum Spanning Tree (MST) algorithm [15]). We define *Immediate-Acceptance Auctions (IAAs)* as mechanisms using only in-queries.

Corollary 1. *An IAA that uses algorithm f is OSP if and only if f is monotone forward greedy.*

We note how all fixed-priority algorithms are OSP but not all OSP mechanism must use a fixed-priority algorithm. In fact, for a fixed-priority algorithm the sufficiency proof alone would go through; the second parameter of the priority functions is only needed for the opposite direction.

Algorithm 2: Reverse greedy algorithm

```
Let b_1, ..., b_n the input bids
P ← F (P is the set of all feasible solutions)
I ← ∅ (I is the set of in agents, i.e., those that cannot be dropped)
A ← N (A is the set of active or in agents)
while |P| > 1 do
    Let i = arg max_{k∈A\I} g_k^{(out)}(b_k, b_{N\A})
    if there are solutions S in P such that i ∉ S then
        Drop from P all the solutions S such that i ∈ S (if any)
        A ← A \ {i}
    else
        I ← I ∪ {i}
Return the only solution in P
```

It is important to compare the result above with [20, Footnote 15], where it is observed how the strategic properties of forward and reverse greedy algorithms are different. The remark applies to auctions where these algorithms are augmented by the so-called threshold payment scheme. Our result shows that there exist alternative OSP payment schemes for this important algorithmic design paradigm.

Deferred-Acceptance Auctions and Reverse Greedy. Another algorithmic approach that can be used to obtain OSP mechanisms is reverse greedy (see Algorithm 2), that is, having an out-priority function that is antimonotone with each agent's type and drops agents accordingly. In its auction format, this is known as Deferred-Acceptance Auctions (DAAs), see, e.g., [20].

Definition 7 (Reverse Greedy). *A reverse greedy algorithm uses functions $g_i^{(out)} : \mathbb{R} \times \mathbb{R}^k \to \mathbb{R}$, $k \leq n-1$, to rank the bids of the players and iteratively excludes the player with highest rank (if feasible) until only one solution is left (cf. Algorithm 2). A reverse greedy algorithm is antimonotone if each $g_i^{(out)}$ is antimonotone in i's private type (i.e., its first argument).*

Similarly to the case of forward greedy, the algorithm need not use $\mathcal{I}$ for *downward-closed* problems (i.e., every subset of a feasible solution is feasible as well). Incidentally, this is the way it is discussed in [7].

Corollary 2 ([20]). *A DAA using algorithm f is OSP if and only if f is anti-monotone reverse greedy.*

Algorithm 3: Two-way greedy algorithm

Let $b_1, \ldots, b_n$ the input bids
$\mathcal{P} \leftarrow \mathcal{F}$ ($\mathcal{P}$ is the set of all feasible solutions)
$\mathcal{I} \leftarrow \emptyset$ ($\mathcal{I}$ is the set of infeasible and in agents)
$\mathcal{A} \leftarrow N$ ($\mathcal{A}$ is the set of all agents that are active or infeasible or in)
while $|\mathcal{P}| > 1$ **do**
  Let $i^{(in)} = \arg\max_{k \in \mathcal{A} \setminus \mathcal{I}} g_k^{(in)}(b_k, \mathbf{b}_{N \setminus \mathcal{A}})$
  Let $i^{(out)} = \arg\max_{k \in \mathcal{A} \setminus \mathcal{I}} g_k^{(out)}(b_k, \mathbf{b}_{N \setminus \mathcal{A}})$
  Let i be the agent corresponding to $\max\left\{g_{i^{(in)}}^{(in)}(b_{i^{(in)}}, \mathbf{b}_{N \setminus \mathcal{A}}), g_{i^{(out)}}^{(out)}(b_{i^{(out)}}, \mathbf{b}_{N \setminus \mathcal{A}})\right\}$
  if $i = i^{(in)} \wedge$ *there are solutions S in $\mathcal{P}$ such that $i \in S$* **then**
    Drop from $\mathcal{P}$ all the solutions S such that $i \notin S$ (if any)
    $\mathcal{A} \leftarrow \mathcal{A} \setminus \{i\}$
  else if $i = i^{(out)} \wedge$ *there are solutions S in $\mathcal{P}$ such that $i \notin S$* **then**
    Drop from $\mathcal{P}$ all the solutions S such that $i \in S$ (if any)
    $\mathcal{A} \leftarrow \mathcal{A} \setminus \{i\}$
  else
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
Return the only solution in $\mathcal{P}$

It is convenient to discuss the relative power of forward and reverse greedy algorithms. We now know from Corollaries 1 and 2 that they are strategically equivalent. Algorithmically, however, there are some differences. There are algorithms and problems, such as, Kruskal algorithm for MST, where we can take the reverse version of forward greedy (e.g., for MST, start with the entire edge set, go through the edges from the most expensive to the cheaper, and remove an edge whenever it does not disconnect the graph) without any consequence to the approximation guarantee. For the minimum spanning tree problem (and, more generally, for finding the minimum-weight basis of a matroid), the reverse greedy algorithm is just as optimal as the forward one. In general (and even for, e.g., bipartite matching), the reverse version of a forward greedy algorithm with good approximation guarantee can be bad [7].

OSP Mechanisms and Two-Way Greedy. We show that the right algorithmic technique for OSP is a suitable combination of forward and reverse greedy.

Definition 8 (Two-way Greedy). *A two-way greedy algorithm uses functions* $g_i^{(in)} : \mathbb{R} \times \mathbb{R}^k \to \mathbb{R}$ *and* $g_i^{(out)} : \mathbb{R} \times \mathbb{R}^k \to \mathbb{R}$, $k \leq n-1$, *for each agent* i, *to rank the bids, and iteratively and greedily includes (if highest priority is defined by* $g^{(in)}$*) or excludes (if highest priority is defined by* $g^{(out)}$*) whenever possible the player with the highest rank until one feasible solution is left (cf. Algorithm 3). A two-way greedy is all-monotone if each* $g_i^{(in)}$ *is monotone in* i*'s private type and* $g_i^{(out)}$ *is antimonotone in* i*'s private type (i.e., their first argument).*

To fully capture the strategic properties of two-way greedy algorithms, we need to define one more property for which we need some background definitions. Consider the total increasing[3] ordering φ of the $2\prod_i |D_i|$ functions[4] $\{g_i^{(\star)}(b, \mathbf{b})\}_{i \in N, b \in D_i, \mathbf{b} \in D_{-i}, \star \in \{in,out\}}$ used by a two-way greedy algorithm. Given $\varphi_\ell = g_i^{(\star)}(b_i, \mathbf{b}_{N \setminus \mathcal{A}_\ell})$, the ℓ-th entry of φ, we let $D_j^{\prec}(\ell)$ ($D_j^{\succ}(\ell)$, respectively) denote the set of types $b \in D_j$ such that $g_j^{(\star)}(b, \mathbf{b}_{N \setminus \mathcal{A}}) > \varphi_\ell$ ($g_j^{(\star)}(b, \mathbf{b}_{N \setminus \mathcal{A}}) < \varphi_\ell$, respectively) with $\mathcal{A} \supseteq \mathcal{A}_\ell$ ($\mathcal{A} \subseteq \mathcal{A}_\ell$, respectively). Moreover, we add b_i (the bid defining φ_ℓ) to $D_i^{\prec}(\ell)$. In words, once in the ordering we reach the ℓ-th entry for a certain agent type and a given "history" (i.e., $\mathbf{b}_{N \setminus \mathcal{A}_\ell}$) with $D_j^{\prec}(\ell)$ we denote all the types that the algorithm has already explored for agent j at this point (that is, for a compatible prior history $\mathbf{b}_{N \setminus \mathcal{A}_\ell}$ with $\mathcal{A} \supseteq \mathcal{A}_\ell$). Similarly, $D_j^{\succ}(\ell)$ denotes those types in D_j that are yet to be considered from this history onwards. Finally, for $d \in \{in, out\}$ we let $\overline{d}$ be a shorthand for the other direction.

Definition 9 (Interleaving Algorithm). *We say that a two-way greedy algorithm is interleaving if for each* i *and* $\mathcal{A} \subseteq N$ *the following occurs. For each* $\varphi_\ell = g_i^{(d)}(b, \mathbf{b}_{N \setminus \mathcal{A}})$ *such that for some* $\mathcal{A}' \subseteq \mathcal{A}$ *it holds* $g_i^{(\overline{d})}(b', \mathbf{b}_{N \setminus \mathcal{A}'}) = \varphi_{\ell'}$ *with* $b' \in D_i^{\succ}(\ell)$ *(and then* $\ell' > \ell$*) we have* $g_i^{(\star)}(x, \mathbf{b}_{N \setminus \mathcal{A}'}) > g_j^{(\star)}(y, \mathbf{b}_{N \setminus \mathcal{A}'})$ *for each* $y \in D_j^{\succ}(\ell')$ *and for all (but at most one)* x *in* $D_i^{\succ}(\ell')$ *(*$\star \in \{in, out\}$*).*

The definition above captures in algorithmic terms the weak interleaving property of extensive-form implementations. Whenever there is a change of direction (from d to $\overline{d}$) for a certain agent i and two compatible histories (cf. condition $\mathcal{A}' \subseteq \mathcal{A}$) then it must be the case that i is revealable and all the other unexplored types (but at most one) *must* be explored next.

Example 1 (Interleaving Algorithm). Consider a setting with three agents, called x, y and z. The *valuation* domain is the same for all the agents and has maximum $t_{\max}$ and minimum $t_{\min}$. Consider the two-way greedy algorithm with the following ordering φ: $g_x^{(in)}(t_{\max}) > g_y^{(in)}(t_{\max}) > g_z^{(out)}(t_{\min}) > g_y^{(out)}(t_{\min}) > \ldots$ (where the second argument is omitted since it is $\emptyset$). Let us focus on $\varphi_2 =$

[3] For notational simplicity, we here assume that there are not ties between the priority functions.

[4] The algorithm must not necessarily have a definition for the priority functions for all the combinations of type/history as some might never get explored. In this case, we set all the undefined entries to sufficiently small (tie-less, for simplicity) values.

$g_y^{(in)}(t_{\max})$. Here $D_x^{\prec}(2)$ and $D_y^{\prec}(2)$ is $\{t_{\max}\}$ (as it has been already considered for both agents) whilst $D_x^{\succ}(2)$ and $D_y^{\succ}(2)$ is equal to the original domain but $t_{\max}$. For z, instead, $D_z^{\prec}(2) = \emptyset$ and $D_z^{\succ}(2)$ is still the original domain. At φ_4 there is a change of direction for agent y. The two-way greedy is interleaving if the domain has only three types (since there are no constraints on the in/out priority for third type in the domain of y). For larger domains, instead, we need to look at the next entries of φ to ascertain whether the algorithm is interleaving or not.

Corollary 3. *A mechanism using algorithm f is OSP if and only if f is an all-monotone interleaving two-way greedy algorithm.*

Given the corollary above, we will henceforth simply say two-way greedy (algorithm) and avoid stating the properties of all-monotonicity and interleaving. We next analyse the approximation guarantee of two-way greedy algorithms, whilst a comparison with forward/reverse greedy is deferred to the full version.

Approximation Guarantee of Two-Way Greedy Algorithms. We next prove that the approximation ratio of two-way greedy algorithms is unbounded in set systems where we want to output the solution with minimum social cost *for some* (minimal) structure of $\mathcal{F}$. Examples include the shortest path problem. For such problems, a two-way greedy algorithm must commit immediately to a solution after the first decision in either Line 10 or 13.

Theorem 7 (Social Cost). *For any $\rho \geq 1$, there exists a set system such that no two-way greedy algorithm returns a ρ-approximation to the optimal social cost, even if there are only two feasible solutions and four agents.*

We now consider the case where the players have valuations and we are interested in maximizing the social welfare. We call this setup where no further assumption on the structure of the feasible solutions in $\mathcal{F}$ can be made, a *restricted knapsack auction* problem.

Theorem 8 (Social Welfare). *There exists a set system for which any two-way greedy algorithm has approximation $\Omega(\sqrt{n})$ to the optimal social welfare, even if there are only two feasible solutions.*

Interestingly the result above uses a so-called *asymmetric instance* [7] where one solution is a singleton and the other is comprised of all the remaining bidders. We now prove that the analysis above is tight at least for three-value domains.

Theorem 9. *There is a $\sqrt{n-1}$-approximate forward greedy algorithm for the asymmetric restricted knapsack auctions, when bidders have a three-value domain $\{t_{\min}, t_{\text{med}}, t_{\max}\}$.*

We now turn our attention to downward-closed set systems for social welfare maximization. This is a generalization of the setting studied in [7], called *knapsack auctions*: There are n bidders and m copies of one item; each bidder has a private valuation v_i to receive at least s_i copies of the item, s_i being public

knowledge. A solution is feasible if the sum of items allocated to bidders is at most m. The objective is social welfare maximization. The authors of [7] give a $O(\ln m)$-approximate DAA/reverse greedy algorithm and prove a lower bound of $\ln^{\tau} m$, for a positive constant τ, *limited to* DAA/reverse greedy. We next show that the upper bound is basically tight for the entire class of OSP mechanisms.

Theorem 10 (Social Welfare Downward-Closed Set Systems). *There is a downward-closed set system for which every two-way greedy algorithm has approximation $\Omega(\sqrt{\ln n})$ to the optimal social welfare.*

5 Conclusions

OSP has attracted lots of interest in computer science and economics, see, e.g., [6,8–11,16,19]. Our work can facilitate the study of this notion of incentive-compatibility for imperfectly rational agents. Just as the characterization of DAAs in terms of reverse greedy [20] has given the first extrinsic reason to study the power and limitations of these algorithms [7,12], we believe that our characterization of OSP in terms of two-way greedy will lead to a better understanding of this algorithmic paradigm. In this work, we only began to investigate their power and much more is left to be done. For example, different optimization problems and objective functions could be considered. Moreover, whilst in general OSP mechanisms do not compose sequentially, see, e.g., [3], we could study under what conditions two-way greedy algorithms compose.

References

1. Ausubel, L.M.: An efficient ascending-bid auction for multiple objects. AER **94**(5), 1452–1475 (2004)
2. Avis, D.: A survey of heuristics for the weighted matching problem. Networks **13**(4), 475–493 (1983)
3. Bade, S., Gonczarowski, Y.: Gibbard-satterthwaite success stories and obvious strategyproofness. In: EC, p. 565 (2017)
4. Borodin, A., Nielsen, M.N., Rackoff, C.: (Incremental) priority algorithms. Algorithmica **37**, 295–326 (2003)
5. Clarkson, K.L.: A modification of the greedy algorithm for vertex cover. Inf. Process. Lett. **16**(1), 23–25 (1983)
6. de Keijzer, B., Kyropoulou, M., Ventre, C.: Obviously strategyproof single-minded combinatorial auctions. In: ICALP, pp. 71:1–71:17 (2020)
7. Dütting, P., Gkatzelis, V., Roughgarden, T.: The performance of deferred-acceptance auctions. Math. Oper. Res. **42**(4), 897–914 (2017)
8. Ferraioli, D., Meier, A., Penna, P., Ventre, C.: Automated optimal OSP mechanisms for set systems. In: Caragiannis, I., Mirrokni, V., Nikolova, E. (eds.) WINE 2019. LNCS, vol. 11920, pp. 171–185. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35389-6_13
9. Ferraioli, D., Meier, A., Penna, P., Ventre, C.: Obviously strategyproof mechanisms for machine scheduling. In: ESA, pp. 46:1–46:15 (2019)

10. Ferraioli, D., Ventre, C.: Probabilistic verification for obviously strategyproof mechanisms. In: IJCAI, pp. 240–246 (2018)
11. Ferraioli, D., Ventre, C.: Approximation guarantee of OSP mechanisms: the case of machine scheduling and facility location. Algorithmica **83**(2), 695–725 (2021)
12. Gkatzelis, V., Markakis, E., Roughgarden, T.: Deferred-acceptance auctions for multiple levels of service. In: EC (2017)
13. Hausmann, D., Korte, B., Jenkyns, T.A.: Worst case analysis of greedy type algorithms for independence systems. In: Padberg, M.W. (ed.) Combinatorial Optimization, pp. 120–131. Springer, Heidelberg (1980). https://doi.org/10.1007/BFb0120891
14. Kagel, J.H., Harstad, R.M., Levin, D.: Information impact and allocation rules in auctions with affiliated private values: a laboratory study. Econometrica **55**(6), 1275–1304 (1987)
15. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. **7**(1), 48–50 (1956)
16. Kyropoulou, M., Ventre, C.: Obviously strategyproof mechanisms without money for scheduling. In: AAMAS, pp. 1574–1581 (2019)
17. Lehmann, D., O'Callaghan, L., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. J. ACM **49**(5), 577–602 (2002)
18. Li, S.: Obviously strategy-proof mechanisms. AER **107**(11), 3257–87 (2017)
19. Mackenzie, A.: A revelation principle for obviously strategy-proof implementation. Games Econ. Behav. **124**, 512–533 (2018)
20. Milgrom, P., Segal, I.: Clock auctions and radio spectrum reallocation. J. Polit. Econ. **128**(1), 1–31 (2020)
21. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V. (eds.) Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
22. Saks, M., Yu, L.: Weak monotonicity suffices for truthfulness on convex domains. In: EC (2005)