2007

Incremental Mechanism Design

Vincent Conitzer
Duke University

Tuomas W. Sandholm
Carnegie Mellon University

Published In
Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)..

Incremental Mechanism Design*

Vincent Conitzer
Duke University
Department of Computer Science
conitzer@cs.duke.edu

Tuomas Sandholm
Carnegie Mellon University
Computer Science Department
sandholm@cs.cmu.edu

Abstract

Mechanism design has traditionally focused almost exclusively on the design of truthful mechanisms. There are several drawbacks to this: 1. in certain settings (*e.g.* voting settings), no desirable strategy-proof mechanisms exist; 2. truthful mechanisms are unable to take advantage of the fact that computationally bounded agents may not be able to find the best manipulation, and 3. when designing mechanisms automatically, this approach leads to constrained optimization problems for which current techniques do not scale to very large instances. In this paper, we suggest an entirely different approach: we start with a naïve (manipulable) mechanism, and incrementally make it *more* strategy-proof over a sequence of iterations.

We give examples of mechanisms that (variants of) our approach generate, including the VCG mechanism in general settings with payments, and the plurality-with-runoff voting rule. We also provide several basic algorithms for automatically executing our approach in general settings. Finally, we discuss how computationally hard it is for agents to find any remaining beneficial manipulation.

1 Introduction

In many multiagent settings, we must choose an outcome based on the preferences of multiple self-interested agents, who will not necessarily report their preferences truthfully if it is not in their best interest to do so. Typical settings in which this occurs include auctions, reverse auctions, exchanges, voting settings, public good settings, resource/task allocation settings, ranking pages on the web [1], *etc.* Research in *mechanism design* studies how to choose outcomes in such a way that good outcomes are obtained even when agents respond to incentives to misreport their preferences (or *manipulate*). For the most part, researchers have focused simply on creating *truthful* (or *strategy-proof*) mechanisms, in which no agent ever has an incentive to misreport. This approach is typically justified by appealing to a result known as the *revelation principle*, which states that for any mechanism that does well in the face of strategic misreporting by agents, there is a truthful mechanism that will perform just as well.

*This material is based upon work supported by the National Science Foundation under ITR grants IIS-0121678 and IIS-0427858, a Sloan Fellowship, and an IBM Ph.D. Fellowship.

The traditional approach to mechanism design has been to try to design good mechanisms that are as general as possible. Probably the best-known general mechanism is the *Vickrey-Clarke-Groves (VCG)* mechanism [16; 4; 10], which chooses the allocation that maximizes the sum of the agents' utilities (the *social welfare*), and makes every agent pay the externality that he[1] imposes on the other agents. This is sufficient to ensure that no individual agent has an incentive to manipulate, but it also has various drawbacks: for example, the surplus payments can, in general, not be redistributed, and the designer may have a different objective than social welfare, *e.g.* she may wish to maximize revenue. Other general mechanisms have their own drawbacks, and there are various impossibility results such as the Gibbard-Satterthwaite theorem [8; 15] that show that certain objectives cannot be achieved by truthful mechanisms.

The lack of a general mechanism that is always satisfactory led to the creation of the field of *automated mechanism design* [5]. Rather than try to design a mechanism that works for a range of settings, the idea is to have a computer automatically compute the optimal mechanism for the specific setting at hand, by solving an optimization problem. A drawback of that approach is that current techniques do not scale to very large instances. This is in part due to the fact that, to ensure strategy-proofness, one must simultaneously decide on the outcome that the mechanism chooses *for every possible input* of revealed preferences, and the strategy-proofness constraints interrelate these decisions.

Another observation that has been made is that in complex settings, it is unreasonable to believe that every agent is endowed with the computational abilities to compute an optimal manipulation. This invalidates the above-mentioned revelation principle, in that restricting attention to truthful mechanisms may in fact come at a cost in the quality of the outcomes that the mechanism produces. Adding to this the observation that in some domains, all strategy-proof mechanisms are unsatisfactory (by the Gibbard-Satterthwaite theorem), it becomes important to be able to design mechanisms that are not strategy-proof. Recent research has already proposed some manipulable mechanisms. There has been work that proposes relaxing the constraint to *approximate* truthfulness (in various senses). Approximately truthful mechanisms can be easier to execute [12; 2], or can circumvent impossiblity results that apply to truthful mechanisms [14; 9]. Other work has studied manipulable mechanisms in which

[1] We will use "she" for the center/designer, and "he" for an agent.

finding a beneficial manipulation is computationally difficult in various senses [3; 13; 6; 7].

In this paper, we introduce a new approach. We start with a naïvely designed mechanism that is not strategy-proof (for example, the mechanism that would be optimal in the absence of strategic behavior), and we attempt to make it *more* strategy-proof. Specifically, the approach systematically identifies situations in which an agent has an incentive to manipulate, and corrects the mechanism to take away this incentive. This is done iteratively, and the mechanism may or may not become (completely) strategy-proof eventually. The final mechanism may depend on the order in which possible manipulations are considered.

One can conceive of this approach as being a computationally more efficient approach to automated mechanism design, insofar as the updates to the mechanism to make it more strategy-proof can be executed automatically (by a computer). Indeed, we will provide algorithms for doing so. It is also possible to think about the results of this approach theoretically, and use them as a guide in "traditional" mechanism design. We will pursue this as well, giving various examples. Finally, we will argue that if the mechanism that the approach produces remains manipulable, then any remaining manipulations will be computationally hard to find.

This approach bears some similarity to how mechanisms are designed in the real world. Real-world mechanisms are often initially naïve, leading to undesirable strategic behavior; once this is recognized, the mechanism is amended to disincent the undesirable behavior. For example, some naïvely designed mechanisms give bidders incentives to postpone submitting their bids until just before the event closes (*i.e.*, sniping); often this is (partially) fixed by adding an *activity rule*, which prevents bidders that do not bid actively early from winning later. As another example, in the 2003 Trading Agent Competition Supply Chain Management (TAC/SCM) game, the rules of the game led the agents to procure most of their components on day 0. This was deemed undesirable, and the designers tried to modify the rules for the 2004 competition to disincent this behavior [11].[2]

As we will see, there are many variants of the approach, each with its own merits. We will not decide which variant is the best in this paper; rather, we will show for a few different variants that they can result in desirable mechanisms.

2 Mechanism design background

In a mechanism design setting, we are given:
• A set of agents N ($|N| = n$);
• A set of outcomes O (here, if payments are used in the setting, an outcome includes information on payments to be made by/to the agents);
• For each agent $i \in N$, a set of types Θ_i (and we denote by $\Theta = \Theta_1 \times \ldots \times \Theta_n$ the set of all type vectors, *i.e.* the set of all possible inputs to the mechanism);
• For each $i \in N$, a utility function $u_i : \Theta_i \times O \rightarrow \mathbb{R}$;[3]
• An objective function $g : \Theta \times O \rightarrow \mathbb{R}$.

For example, in a single-item auction, N is the set of bidders; $O = S \times \Pi$, where S is the set of all possible allocations of the item (one for each bidder, plus potentially one allocation where no bidder wins), and Π is the set of all possible vectors $\langle \pi_1, \ldots, \pi_n \rangle$ of payments to be made by the agents (*e.g.*, $\Pi = \mathbb{R}^n$); assuming no allocative externalities (that is, it does not matter to a bidder which other bidder wins the item if the bidder does not win himself), Θ_i is the set of possible valuations that the bidder may have for the item (for example, $\Theta_i = \mathbb{R}^{\geq 0}$); the utility function u_i is given by: $u_i(\theta_i, (s, \langle \pi_1, \ldots, \pi_n \rangle)) = \theta_i - \pi_i$ if s is the outcome in which i wins the item, and $u_i(\theta_i, (s, \langle \pi_1, \ldots, \pi_n \rangle)) = -\pi_i$ otherwise. (In situations in which a type consists of a single value, we will typically use v_i rather than θ_i for the type.)[4]

A (deterministic) *mechanism* consists of a function $M : \Theta \rightarrow O$, specifying an outcome for every vector of (reported) types.[5] Given a mechanism M, a *beneficial manipulation*[6] consists of an agent $i \in N$, a type vector $\langle \theta_1, \ldots, \theta_n \rangle \in \Theta$, and an alternative type report $\hat{\theta}_i$ for agent i such that $u_i(\theta_i, M(\langle \theta_1, \ldots, \theta_n \rangle)) < u_i(\theta_i, M(\langle \theta_1, \ldots, \theta_{i-1}, \hat{\theta}_i, \theta_{i+1}, \ldots, \theta_n \rangle))$. In this case we say that i manipulates *from* $\langle \theta_1, \ldots, \theta_n \rangle$ *into* $\langle \theta_1, \ldots, \theta_{i-1}, \hat{\theta}_i, \theta_{i+1}, \ldots, \theta_n \rangle$. A mechanism is *strategy-proof* or *(dominant-strategies) incentive compatible* if there are no beneficial manipulations. (We will not consider Bayes-Nash equilibrium incentive compatibility here.)

In settings with payments, we enforce an *ex-post individual rationality* constraint: we cannot make an agent worse off than he would have been if he had not participated. That is, we cannot charge an agent more than he reported the outcome (disregarding payments) was worth to him.

3 Our approach and techniques

In this section, we explain the approach and techniques that we consider in this paper. We recall that our goal is not to (immediately) design a strategy-proof mechanism; rather, we start with some manipulable mechanism, and attempt to incrementally make it "more" strategy-proof. Thus, the basic template of our approach is as follows:
1. Start with some (manipulable) mechanism M;
2. Find some set F of manipulations (where a manipulation is given by an agent $i \in N$, a type vector $\langle \theta_1, \ldots, \theta_n \rangle$, and an alternative type report $\hat{\theta}_i$ for agent i);
3. If possible, change the mechanism M to prevent (many of) these manipulations from being beneficial;
4. Repeat from step 2 until termination.

This is merely a template; at each one of the steps, something remains to be filled in. Which initial mechanism do we

[2] Interestingly, these *ad-hoc* modifications failed to prevent the behavior, and even an extreme modification during the 2004 competition failed. Later research suggests that in fact all reasonable settings for a key parameter would have failed [17].

[3] The utility function is parameterized by type; while the u_i are common knowledge, the types encode (private) preferences.

[4] In general, we may have additional information, such as a prior over the types, but we will not use this information in this paper.

[5] In general, a mechanism may be randomized, specifying distributions over outcomes, but we will not consider this in this paper.

[6] "Beneficial" here means beneficial to the manipulating agent.

choose in step 1? Which set of manipulations do we consider in step 2? How do we "fix" the mechanism in step 3 to prevent these manipulations? And how do we decide to terminate in step 4? In this paper, we will not resolve what is the best way to fill in these blanks (it seems unlikely that there is a single, universal best way), but rather we will provide a few instantiations of the technique, illustrate them with examples, and show some interesting properties.

One natural way of instantiating step 1 is to choose a *naïvely optimal* mechanism, that is, a mechanism that would give the highest objective value for each type vector *if* every agent would always reveal his type truthfully. For instance, if we wish to maximize social welfare, we simply always choose an outcome that maximizes social welfare for the reported types; if we wish to maximize revenue, we choose an outcome that maximizes social welfare for the reported types, and make each agent pay his entire valuation.

In step 2, there are many possible options: we can choose the set of *all* manipulations; the set of all manipulations for a single agent; the set of all manipulations from or to a particular type or type vector; or just a single manipulation. The structure of the specific setting under consideration may also make certain manipulations more "natural" than others; we can discover which manipulations are more natural by intuition, by hiring agents to act in test runs of the mechanism, by running algorithms that find manipulations, *etc.* Which set of manipulations we choose will affect the difficulty of step 3.

Step 3 is the most complex step. Let us first consider the case where we are only trying to prevent a single manipulation, from $\theta = \langle \theta_1, \ldots, \theta_n \rangle$ to $\theta' = \langle \theta_1, \ldots, \theta_{i-1}, \hat{\theta}_i, \theta_{i+1}, \ldots, \theta_n \rangle$. We can make this manipulation undesirable in one of three ways: **(a)** make the outcome that M selects for θ more desirable for agent i (when he has type θ_i), **(b)** make the outcome that M selects for θ' less desirable for agent i (when he has type θ_i), or **(c)** a combination of the two. We will focus on **(a)** in this paper. There may be multiple ways to make the outcome that M selects for θ sufficiently desirable to prevent the manipulation; a natural way to select from among these outcomes is to choose the one that maximizes the designer's original objective. Note that these modifications may introduce other beneficial manipulations.

When we are trying to prevent a set of manipulations, we are confronted with an additional issue: after we have prevented one manipulation in the set, we may reintroduce the incentive for this manipulation when we try to prevent another manipulation. Resolving this would require solving a potentially large constrained optimization problem, constituting an approach similar to standard automated mechanism design—reintroducing some of the scalability problems that we wish to avoid. Therefore, when addressing the manipulations from one type vector, we will simply act as if we will not change the outcomes for any other type vector.

Formally, for this particular instantiation of our approach, if M is the mechanism at the beginning of the iteration and M' is the mechanism at the end of the iteration (after the update), and F is the set of manipulations under consideration, we have $M'(\theta) \in \arg\max_{o \in O(M,\theta,F)} g(\theta, o)$ (here, $\theta = \langle \theta_1, \ldots, \theta_n \rangle$), where $O(M, \theta, F) \subseteq O$ is the set of all outcomes o such that for any beneficial manipulation $(i, \hat{\theta}_i)$ (with $(i, \theta, \hat{\theta}_i) \in F$), $u_i(\theta_i, o) \geq u_i(\theta_i, M(\langle \theta_1, \ldots, \theta_{i-1}, \hat{\theta}_i, \theta_{i+1}, \ldots, \theta_n \rangle))$. It may happen that $O(M, \theta, F) = \emptyset$ (no outcome will prevent all manipulations). In this case, there are various ways in which we can proceed. One is not to update the outcome at all, *i.e.* set $M'(\theta) = M(\theta)$. Another is to minimize the number of agents that will have an incentive to manipulate from θ after the change, that is, to choose $M'(\theta) \in \arg\min_{o \in O} |\{i \in N : (\exists (i, \theta, \hat{\theta}_i) \in F : u_i(\theta_i, o) < u_i(\theta_i, M(\langle \theta_1, \ldots, \theta_{i-1}, \hat{\theta}_i, \theta_{i+1}, \ldots, \theta_n \rangle)))\}|$ (and ties can be broken to maximize the objective g).

Many other variants are possible. For example, instead of choosing from the set of all possible outcomes O when we update the outcome of the mechanism for some type vector θ, we can limit ourselves to the set of all outcomes that would result from some beneficial manipulation in F from θ—that is, the set $\{o \in O : ((\exists (i, \hat{\theta}_i) : (i, \theta, \hat{\theta}_i) \in F) : o = M(\langle \theta_1, \ldots, \theta_{i-1}, \hat{\theta}_i, \theta_{i+1}, \ldots, \theta_n \rangle))\}$—in addition to the current outcome $M(\theta)$. The motivation is that rather than consider all possible outcomes every time, we may wish to simplify our job by considering only the ones that cause the failure of strategy-proofness in the first place. We next present examples of some of the above-mentioned variants.

4 Instantiating the methodology

In this section, we illustrate the potential benefits of the approach by exhibiting mechanisms that it can produce in various standard mechanism design settings. We will demonstrate a setting in which the approach ends up producing a strategy-proof mechanism, as well as a setting in which the produced mechanism is still vulnerable to manipulation (but in some sense "more" strategy-proof than naïve mechanisms). (A third setting that we studied—deciding on whether to produce a public good—is omitted due to space constraint.) We emphasize that our goal in this section is not to come up with spectacularly novel mechanisms, but rather to show that the approach advocated in this paper produces sensible results. Therefore, for now, we will consider the approach successful if it produces a well-known mechanism. In future research, we hope to use the technique to help us design novel mechanisms as well.[7] We do emphasize, however, that although the mechanisms that the approach eventually produces were already known to *us*, the *approach* simply follows local updating rules without any knowledge of what the final mechanism should be. In other words, the algorithm is not even given a hint of what the final mechanism should look like.

4.1 Settings with payments

In this subsection, we show the following result: in general preference aggregation settings in which the agents can make payments (*e.g.* combinatorial auctions), (one variant of) our technique yields the VCG mechanism after a single iteration. We recall that the VCG mechanism chooses an outcome that

[7] Certainly, if we apply the approach to a previously unstudied mechanism design domain, it *will* produce a novel mechanism. However, it would be difficult to evaluate the quality of such a mechanism, since there would be nothing to compare the mechanism to.

maximizes social welfare (not counting payments), and imposes the following tax on an agent: consider the total utility (not counting payments) of the other agents given the chosen outcome, and subtract this from the total utility (not counting payments) that the other agents *would have obtained* if the given agent's preferences had been ignored in choosing the outcome. Specifically, we will consider the following variant of our technique (perhaps the most basic one):

• Our objective g is to try maximize some (say, linear) combination of allocative social welfare (*i.e.* social welfare not taking payments into account) and revenue. (It does not matter what the combination is.)

• The set F of manipulations that we consider is that of all possible misreports (by any single agent).

• We try to prevent manipulations according to **(a)** above (for a type vector from which there is a beneficial manipulation, make its outcome desirable enough to the manipulating agents to prevent the manipulation). Among outcomes that achieve this, we choose one maximizing the objective g.

We will use the term "allocation" to refer to the part of the outcome that does not concern payments, even though the result is not restricted to allocation settings such as auctions. Also, we will refer to the utility that agent i with type θ_i gets from allocation s (not including payments) as $u_i(\theta_i, s)$. The following simple observation shows that the naïvely optimal mechanism is the *first-price* mechanism, which chooses an allocation that maximizes social welfare, and makes every agent pay his valuation for the allocation.

Observation 1 *The first-price mechanism naïvely maximizes both revenue and allocative social welfare.*

Proof: That the mechanism (naïvely) maximizes allocative social welfare is clear. Moreover, due to the individual rationality constraint, we can never extract more than the allocative social welfare; and the first-price mechanism (naïvely) extracts all the allocative social welfare, for an outcome that (naïvely) maximizes allocative social welfare. ■

Before showing the main result of this subsection, we first characterize optimal manipulations under the first-price mechanism.

Lemma 1 *The following is an optimal manipulation $\hat{\theta}_i$ from $\theta \in \Theta$ for agent i under the first-price mechanism:*

• *for the allocation s^* that would be chosen under the first-price mechanism for θ, report a value equal to i's VCG payment under the true valuations ($u(\hat{\theta}_i(s^*)) = VCG_i(\theta_i, \theta_{-i})$);*

• *for any other allocation $s \neq s^*$, report a valuation of* 0.[8]

The utility of this manipulation is $u(\theta_i, s^*) - VCG_i(\theta_i, \theta_{-i})$. (This assumes ties will be broken in favor of allocation s^*.)

[8]There may be constraints on the reported utility function that prevent this—for example, in a (combinatorial) auction, perhaps only monotone valuations are allowed (winning more items never hurts an agent). If so, the agent should report valuations for these outcomes that are as small as possible, which will still lead to s^* being chosen.

Without the tie-breaking assumption, the lemma does not hold: for example, in a single-item first-price auction, bidding exactly the second price for the item is not an optimal manipulation for the bidder with the highest valuation if the tie is broken in favor of the other bidder. However, increasing the bid by any amount will guarantee that the item is won (and in general, increasing the value for s^* by any amount will guarantee that outcome).

Proof: First, we show that this manipulation will still result in s^* being chosen. Suppose that allocation $s \neq s^*$ is chosen instead. Given the tie-breaking assumption, it follows that $\sum_{j \neq i} u_j(\theta_j, s) > u_i(\hat{\theta}_i, s^*) + \sum_{j \neq i} u_j(\theta_j, s^*)$, or equivalently, $VCG_i(\theta_i, \theta_{-i}) < \sum_{j \neq i} u_j(\theta_j, s) - u_j(\theta_j, s^*)$. However, by definition, $VCG_i(\theta_i, \theta_{-i}) = \max_{s^{**}} \sum_{j \neq i} u_j(\theta_j, s^{**}) - u_j(\theta_j, s^*) \geq \sum_{j \neq i} u_j(\theta_j, s) - u_j(\theta_j, s^*)$, so we have the desired contradiction. It follows that agent i's utility under the manipulation is $u_i(\theta_i, s^*) - VCG_i(\theta_i, \theta_{-i})$.

Next, we show that agent i cannot obtain a higher utility with any other manipulation. Suppose that manipulation $\hat{\theta}_i$ results in allocation s being chosen. Because utilities cannot be negative under truthful reporting, it follows that $u_i(\hat{\theta}_i, s) + \sum_{j \neq i} u_j(\theta_j, s) \geq \max_{s^{**}} \sum_{j \neq i} u_j(\theta_j, s^{**})$. Using the fact that $VCG_i(\theta_i, \theta_{-i}) = \max_{s^{**}} \sum_{j \neq i} u_j(\theta_j, s^{**}) - u_j(\theta_j, s^*)$, we can rewrite the previous inequality as $u_i(\hat{\theta}_i, s) + \sum_{j \neq i} u_j(\theta_j, s) \geq VCG_i(\theta_i, \theta_{-i}) + \sum_{j \neq i} u_j(\theta_j, s^*)$, or equivalently $u_i(\hat{\theta}_i, s) \geq VCG_i(\theta_i, \theta_{-i}) + \sum_{j \neq i} u_j(\theta_j, s^*) - u_j(\theta_j, s)$. Because $\sum_j u_j(\theta_j, s^*) \geq \sum_j u_j(\theta_j, s)$, we can rewrite the previous inequality as $u_i(\hat{\theta}_i, s) \geq VCG_i(\theta_i, \theta_{-i}) - u_i(\theta_i, s^*) + u_i(\theta_i, s) + \sum_j u_j(\theta_j, s^*) - u_j(\theta_j, s) \geq VCG_i(\theta_i, \theta_{-i}) - u_i(\theta_i, s^*) + u_i(\theta_i, s)$, or equivalently, $u_i(\theta_i, s) - u_i(\hat{\theta}_i, s) \leq u_i(\theta_i, s^*) - VCG_i(\theta_i, \theta_{-i})$, as was to be shown. ■

Theorem 1 *Under the variant of our approach described above, the mechanism resulting after a single iteration is the VCG mechanism.*

Proof: By Observation 1, the naïvely optimal mechanism is the first-price mechanism. When updating the outcome for θ, by Lemma 1, each agent i must receive a utility of at least $u_i(\theta_i, s^*) - VCG_i(\theta_i, \theta_{-i})$, where s^* is the allocation that maximizes allocative social welfare for type vector θ. One way of achieving this is to choose allocation s^*, and to charge agent i exactly $VCG_i(\theta_i, \theta_{-i})$—that is, simply run the VCG mechanism. Clearly this maximizes allocative social welfare. But, under the constraints on the agents' utilities, it also maximizes revenue, for the following reason. For any allocation s, the most revenue that we can hope to extract is the allocative social welfare of s, that is, $\sum_i u_i(\theta_i, s)$, minus the sum of the utilities that we must guarantee the agents, that

is, $\sum_i u_i(\theta_i, s^*) - VCG_i(\theta_i, \theta_{-i})$. Because $s = s^*$ maximizes $\sum_i u_i(\theta_i, s)$, this means that the most revenue we can hope to extract is $\sum_i VCG_i(\theta_i, \theta_{-i})$, and the VCG mechanism achieves this. ■

4.2 Ordinal preferences

In this subsection, we address voting (social choice) settings. In such a setting, there is a set of outcomes (also known as *candidates* or *alternatives*) and a set of agents (also known as *voters*), and every agent i's type is a complete ranking $\succ_i$ over the candidates. (We do not need to specify numerical utilities here.) The mechanism (or *voting rule*) takes as input the agents' type reports (or *votes*), consisting of complete rankings of the candidates, and chooses an outcome.

The most commonly used voting rule is the *plurality* rule, in which we only consider every voter's highest-ranked candidate, and the winner is simply the candidate with the highest number of votes ranking it first (its *plurality score*). The plurality rule is very manipulable: a voter voting for a candidate that is not winning may prefer to attempt to get the candidate that currently has the second-highest plurality score to win, by voting for that candidate instead. In the real world, one common way of "fixing" this is to add a runoff round, resulting in the *plurality-with-runoff* rule. Under this rule, we take the two candidates with the highest plurality scores, and declare as the winner the one that is ranked higher by more voters. By the Gibbard-Satterthwaite theorem, this is still not a strategy-proof mechanism (it is neither dictatorial nor does it preclude any candidate from winning)—for example, a voter may change his vote to change which candidates are in the runoff. Still, the plurality with runoff rule is, in an intuitive sense, "less" manipulable than the plurality rule (and certainly more desirable than a strategy-proof rule, since a strategy-proof rule would either be dictatorial or preclude some candidate from winning).

In this subsection, we will show that the following variant of our approach will produce the plurality-with-runoff rule when starting with the plurality rule as the initial mechanism.
• The set F consists of all manipulations in which a voter changes which candidate he ranks first.
• We try to prevent manipulations as follows: for a type (vote) vector from which there is a beneficial manipulation, consider all the outcomes that may result from such a manipulation (in addition to the current outcome), and choose as the new outcome the one that minimizes the number of agents that still have an incentive to manipulate from this vote vector.
• We will change the outcome for each vote vector at most once (but we will have multiple iterations, for vote vectors whose outcome did not change in earlier iterations).

We are now ready to present the result. (The remaining proofs are omitted due to space constraint.)

Theorem 2 *For a given type vector θ, suppose that candidate b is ranked first the most often, and a is ranked first the second most often ($s(b) > s(a) > \ldots$, where $s(o)$ is the number of times o is ranked first). Moreover, suppose that the number of votes that prefers a to b is greater than or equal to the number of votes that prefers b to a. Then, starting with the plurality rule, after exactly $s(b) - s(a)$ iterations of the approach described above, the outcome for θ changes for the first time, to a (the outcome of the plurality with runoff rule).*[9]

5 Computing the mechanism's outcomes

In this section, we discuss how to automatically compute the outcomes of the mechanisms that are generated by this approach in general. It will be convenient to think about settings in which the set of possible type vectors is finite (so that the mechanism can be represented as a finite table), although these techniques can be extended to (some) infinite settings as well. (At the very least, types can be grouped together into a finite number; for specific settings, something better can often be done.) One potential upside relative to standard automated mechanism design techniques is that we do not need to compute the entire mechanism (the outcomes for all type vectors); rather, we only need to compute the outcome for the type vector that is actually reported.

Let M_0 denote the (naïve) mechanism from which we start, and let M_t denote the mechanism after t iterations. Let F_t denote the set of beneficial manipulations that we are considering (and are trying to prevent) in the tth iteration. Thus, M_t is a function of F_t and M_{t-1}. What this function is depends on the specific variant of the approach that we are using. When we try to prevent manipulations by making the outcome for the type vector from which the agent is manipulating more desirable for that agent, we can be more specific, and say that, for type vector θ, $M_t(\theta)$ is a function of the subset $F_t^\theta \subseteq F_t$ that consists of manipulations that start from θ, and of the outcomes that M_{t-1} selects on the subset of type vectors that would result from a manipulation in F_t^θ. Thus, to compute the outcome that M_t produces on θ, we only need to consider the outcomes that M_{t-1} chooses for type vectors *that differ from θ in at most one type* (and possibly even fewer, if F_t^θ does not consider all possible manipulations). As such, we need to consider M_{t-1}'s outcomes on at most $\sum_{i=1}^{n} |\Theta_i|$ type vectors to compute $M_t(\theta)$ (for any given θ), which is much smaller than the set of all type vectors ($\prod_{i=1}^{n} |\Theta_i|$). Of course, to compute $M_{t-1}(\theta')$ for some type vector θ', we need to consider M_{t-2}'s outcomes on up to $\sum_{i=1}^{n} |\Theta_i|$ type vectors, *etc.*

Because of this, a simple recursive approach for computing $M_t(\theta)$ for some θ will require $O((\sum_{i=1}^{n} |\Theta_i|)^t)$ time. This approach may, however, spend a significant amount of time recomputing values $M_j(\theta')$ many times. Another approach is to use dynamic programming, computing and storing mechanism M_{j-1}'s outcomes on *all* type vectors before proceeding to compute outcomes for M_j. This approach will require $O(t \cdot (\prod_{i=1}^{n} |\Theta_i|) \cdot (\sum_{i=1}^{n} |\Theta_i|))$ time (for every iteration, for every type vector, we must investigate all possible manipula-

[9] This is assuming that ties in the plurality rule are broken in favor of a; otherwise, one more iteration is needed. (Some assumption on tie-breaking must always be made for voting rules.)

tions). We note that when we use this approach, we may as well compute the entire mechanism M_t (we already have to compute the entire mechanism M_{t-1}). If n is large and t is small, the recursive approach is more efficient; if n is small and t is large, the dynamic programming approach is more efficient. We can gain the benefits of both by using the recursive approach and storing the outcomes that we compute in the process, so that we need not recompute them.

All of this is for fully general (finite) domains; it is likely that these techniques can be sped up considerably for specific domains. Moreover, as we have already seen, some domains can simply be solved analytically.

6 Computational hardness of manipulation

We have demonstrated that our approach can change naïve mechanisms into mechanisms that are less (sometimes not at all) manipulable. In this section, we will argue that in addition, if the mechanism remains manipulable, *the remaining manipulations are computationally difficult to find*. This is especially valuable because, as we argued earlier, if it is too hard to discover beneficial manipulations, the revelation principle ceases to hold, and a manipulable mechanism can sometimes actually outperform all truthful mechanisms.

We first give an informal, but general, argument for the claim that any manipulations that remain after a large number of iterations of our approach are hard to find. Suppose that the only knowledge that an agent has about the mechanism is the variant of our approach by which the designer obtains it (the initial naïve mechanism, the manipulations that the designer considers, how she tries to eliminate these opportunities for manipulations, how many iterations she performs, *etc.*). Given this, the most natural algorithm for an agent to find a beneficial manipulation is to simulate our approach for the relevant type vectors, perhaps using the algorithms presented earlier. However, this approach to manipulation is computationally infeasible if the agent does not have the computational capabilities to simulate as many iterations as the designer will actually perform.

Unfortunately, this informal argument fails if the agent actually has greater computational abilities or better algorithms than the designer. However, it turns out that if we allow for *random* updates to the mechanism, then we can prove hardness of manipulation in a formal, complexity-theoretic sense.

So far, we have only discussed updating the mechanism in a deterministic fashion. When the mechanism is updated deterministically, any agent that is computationally powerful enough to simulate this updating process can determine the outcome that the mechanism will choose, for any vector of revealed types. Hence, that agent can evaluate whether he would benefit from misrepresenting his preferences. However, this is not the case if we add random choices to our approach (and the agents are not told about the random choices until after they have reported their types). In fact, we can prove the following result. (As in most previous work on hardness of manipulation, this is only a worst-case notion of hardness, which may not prevent manipulation in all cases.)

Theorem 3 *When the updates to the mechanism are chosen randomly, evaluating whether there exists a manipulation that increases an agent's expected utility is* #P*-hard.*

7 Discussion

While we have given a framework, and a portfolio of techniques within that framework, for making mechanisms more strategy-proof, and illustrated their usefulness with examples, we have not yet integrated the techniques into a single, comprehensive approach. This suggests some important questions for future research. Is there a single, general method that obtains all of the benefits of the individual techniques that we have described (possibly by making use of these techniques as subcomponents)? If not, can we provide some guidance as to which techniques are likely to work best in a given setting? Another direction for future research is to consider other types of manipulation, such as false-name bidding [18].

References

[1] Alon Altman and Moshe Tennenholtz. Ranking systems: The PageRank axioms. *ACM-EC*, 2005.

[2] Aaron Archer, Christos Papadimitriou, K Tawar, and Eva Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. *SODA*, 2003.

[3] John Bartholdi, III and James Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354, 1991.

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

[5] Vincent Conitzer and Tuomas Sandholm. Complexity of mechanism design. *UAI*, pages 103–110, 2002.

[6] Vincent Conitzer and Tuomas Sandholm. Universal voting protocol tweaks to make manipulation hard. *IJCAI*, 2003.

[7] Boi Faltings and Quang Huy Nguyen. Multi-agent coordination using local search. *IJCAI*, 2005.

[8] Allan Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.

[9] Andrew Goldberg and Jason Hartline. Envy-free auctions for digital goods. *ACM-EC*, pages 29–35, 2003.

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

[11] Christopher Kiekintveld, Yevgeniy Vorobeychik, and Michael Wellman. An analysis of the 2004 supply chain management trading agent competition. *IJCAI-05 Workshop on Trading Agent Design and Analysis*, 2005.

[12] Anshul Kothari, David Parkes, and Subhash Suri. Approximately-strategyproof and tractable multi-unit auctions. *ACM-EC*, pages 166–175, 2003.

[13] Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. *ACM-EC*, pages 242–252, 2000.

[14] David Parkes, Jayant Kalagnanam, and Marta Eso. Achieving budget-balance with Vickrey-based payment schemes in exchanges. *IJCAI*, pages 1161–1168, 2001.

[15] Mark Satterthwaite. Strategy-proofness and Arrow's conditions: existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.

[16] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[17] Yevgeniy Vorobeychik, Christopher Kiekintveld, and Michael Wellman. Empirical mechanism design: Methods, with application to a supply chain scenario. *ACM-EC*, 2006.

[18] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. Robust combinatorial auction protocol against false-name bids. *Artificial Intelligence*, 130(2), 2004.