Chapter 10

Automated Mechanism Design for Bounded Agents

A long-term goal of this research is to combine automated mechanism design and mechanism design for bounded agents, so that mechanisms that take advantage of the agents' limited computational capacities are automatically designed. However, the approaches that we have seen in previous chapters cannot be straightforwardly combined to achieve this, for several reasons.

As we saw in Chapter 6, the work on automated mechanism design (ours as well as others') so far has restricted itself to producing truthful mechanisms, and appealed to the revelation principle to justify this. The advantage of this restriction is that it is easy to evaluate the quality of a truthful mechanism, because the agents' behavior is perfectly predictable (they will tell the truth). This is what allows us to come up with a clear formulation of the optimization problem.

However, settings in which the bounded rationality of agents can be exploited are necessarily ones in which the revelation principle does *not* meaningfully apply. Specifically, a mechanism that is truthful (in the sense that no manipulation can be beneficial) can never exploit the agents' bounded rationality, because in such a mechanism agents would reveal the truth regardless of their computational sophistication. It follows that we must extend the search space for automated mechanism design to include non-truthful mechanisms.[1] Finding an optimal mechanism in this extended search space would require us to have some method for evaluating the quality of non-truthful mechanisms, which needs to be based on a model of how the agents behave in non-truthful mechanisms. Moreover, this model *must* take into account the agents' bounded rationality: if we assume that the agents will play optimally (in a game-theoretic sense), then by the revelation principle there is still never a reason to prefer non-truthful mechanisms over truthful mechanisms.[2]

[1]Another aspect of mechanism design for bounded agents is that we may not wish to immediately ask each agent to provide all of its preferences, as these may be difficult for the agent to compute. Rather, we could consider *multistage* mechanisms that selectively query the agent only for the needed preferences. These mechanisms can still be truthful in the sense that it is always strategically optimal to answer queries truthfully. However, as explained before, the efficient elicitation of agents' preferences is a topic that is orthogonal to this dissertation.

[2]Some non-truthful mechanisms may be just as good as truthful mechanisms when agents behave in a game-theoretically optimal way, so it is possible that our search would fortuitously return a non-truthful mechanism. However, if this happens, there is still no guarantee that this non-truthful mechanism will perform better than the best truthful mechanism when agents are actually bounded, for one of two reasons. First, it may be easy to act optimally in this particular

Creating a good model of the behavior of boundedly rational agents for this purpose is by no means easy. For one, it is difficult to guarantee that agents will not adapt their reasoning algorithms to the specific mechanism that they face. The results on hardness of manipulation in Chapter 8 avoided this difficulty, by using the standard complexity-theoretic approach of showing that there are infinite *families* of instances that are hard. This was necessary because in standard formulations of complexity theory, any *individual* instance of a problem is easy to solve, for example by the algorithm that has the solution to that particular instance precomputed. Unfortunately, the purpose of automated mechanism design is precisely to design a mechanism *for the instance at hand* only! Thus we cannot refer to hardness over infinite classes of instances for this purpose.

These difficulties prevent us from formulating automated mechanism design for bounded agents as a clean optimization problem. Nevertheless, in Section 10.1, we do propose a more heuristic approach by which mechanisms for bounded agents can be designed automatically [Conitzer and Sandholm, 2006e]. In light of the issues discussed above, it should not come as a surprise that this approach is very different from the approaches described earlier in this dissertation. Rather than optimizing the entire mechanism in a single step (as in Chapter 6), the idea is to incrementally make the mechanism *more* strategy-proof over time, by finding potential beneficial manipulations, and changing outcomes locally so that these manipulations are no longer beneficial. Computationally, this is a much more scalable approach. The mechanism may eventually become (completely) strategy-proof, in which case it will not take advantage of agents' bounded rationality; however, it may be that some manipulations remain in the end. Intuitively, one should expect such remaining manipulations to be more difficult to discover for the agents, as the algorithm for designing the mechanism has not discovered them yet. Thus, instead of using a complexity-theoretic argument as in Chapter 8, here the argument for hardness of manipulation depends on the agents not being able to "outcompute" the designer. (Nevertheless, we will also give a complexity-theoretic argument.)

10.1 Incrementally making mechanisms more strategy-proof

In the approach that we propose in this section, 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 locally to take away this incentive. This is done iteratively, and the mechanism may or may not become (completely) strategy-proof eventually.

One can conceive of this as being a new 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. (These algorithms are computationally much more efficient than the optimization algorithms proposed in Chapter 6, because to ensure strategy-proofness, those algorithms had to simultaneously decide on the outcome that the mechanism chooses *for every possible input* of revealed preferences, and the strategy-proofness constraints interrelated these decisions.) It is also possible to think about the results of this approach theoretically, and use them as a guide in more "traditional" mechanism design. We pursue this as well,

non-truthful mechanism. Second (worse), it may the case that it is not easy, and that this difficulty actually leads the agents to act in such a way that *worse* outcomes are obtained.

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 somehow 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 [Kiekintveld *et al.*, 2005].[3]

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 dissertation; rather, we will show for a few different variants that they can result in desirable mechanisms.

10.1.1 Definitions

In this chapter, we will consider payments (if they are possible) to be part of the outcome. Because of this, we can identify a mechanism with its outcome selection function. Given a mechanism $M : \Theta \rightarrow O$ mapping type vectors to outcomes, a *beneficial manipulation*[4] consists of an agent i, 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* $\langle\theta_1, \ldots, \theta_{i-1}, \hat{\theta}_i, \theta_{i+1}, \ldots, \theta_n\rangle$. We note that a mechanism is strategy-proof or (dominant-strategies) incentive compatible if and only if there are no beneficial manipulations. (We will not consider Bayes-Nash equilibrium incentive compatibility in this chapter.)

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

10.1.2 Our approach and techniques

In this subsection, we explain the approach and techniques that we consider in this chapter. 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;

[3]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 [Vorobeychik *et al.*, 2006].

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

2. Find some set F of manipulations (where a manipulation is given by an agent i, 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 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 dissertation, 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. Which option 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. For the most part, we will focus on **(a)** in this chapter. 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 any one of these modifications may introduce other beneficial manipulations.)

When we are trying to prevent a set of manipulations, we are confronted with an additional problem: after we have prevented one manipulation in the set, we may reintroduce the incentive for this manipulation when we try to prevent another manipulation. As a simple example, suppose that we are selling a single item to a single bidder, who may value the item at 1, 2, or 3. Suppose that we start with the (naïve) mechanism in which we always sell the item to the bidder at the bid that he places (if he bids $x \in \{1, 2, 3\}$, we sell the item to him at price $\pi(x) = x$). Of course, the bidder has an incentive to shade his valuation. Now suppose that (for some reason) in step 2 we choose the set of the following two beneficial manipulations: report 2 when the true value is 3, and report 1 when the true value is 2. Also suppose that we take approach **(a)** above (for a type vector from which there is a beneficial manipulation, make its outcome more desirable to the manipulating agent). We can fix the first manipulation by setting $\pi(3) = 2$, but then if we fix the second manipulation by setting $\pi(2) = 1$, the incentive to report 2 when the true value is 3 returns. This can be prevented by updating all of the type vectors simultaneously in such a way that none of the manipulations remain beneficial: in the example, this would lead us to find that we should set $\pi(3) = \pi(2) = 1$. However, in general settings, this may require solving a potentially large constrained optimization problem, which would constitute an approach similar to standard automated mechanism design—reintroducing some of the scalability problems that we wish to avoid.[5] Instead, we will be less ambitious: when addressing the manipulations from one type vector, we will act as if we will not change the outcomes for any other type vector. Thus, in the example above, we will indeed choose $\pi(3) = 2$, $\pi(2) = 1$. (Of course, if we had included the manipulation of reporting 1 when the true value is 3, we would set $\pi(3) = \pi(2) = 1$, and there would be no problem. This is effectively what will happen in some of the examples that we will give later.)

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 $M'(\theta) \in \arg\min_{o \in O} |\{i : (\exists (i, \theta, \hat{\theta}_i) \in F : u_i(\theta_i, o) < u_i(\theta_i, M(\langle\theta_1, \ldots, \theta_{i-1}, \hat{\theta}_i, \theta_{i+1}, \ldots, \theta_n\rangle)))\}|$ (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 for this is that rather than have to 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 may, however, get better results by considering all outcomes.)

In the last few paragraphs, we have been focusing on approach **(a)** above (for a type vector from which there is a beneficial manipulation, make its outcome more desirable to the manipulating agent); approach **(b)** (for a type vector *into* which there is a beneficial manipulation, make its outcome *less* desirable to the manipulating agent) can be instantiated with similar techniques. For example, we can redefine $O(M, \theta, F) \subseteq O$ as the set of all outcomes o such that for any manipulation in F *into* θ, choosing $M'(\theta) = o$ prevents this manipulation from being beneficial.

Next, we present examples of all of the above-mentioned variants.

10.1.3 Instantiating the methodology

In this subsection, we illustrate the potential benefits of the approach by exhibiting mechanisms that it can produce in various standard mechanism design settings. We will demonstrate settings in

[5]Although, if the set of manipulations that we are considering is small, this approach may still scale better than standard automated mechanism design (in which *all* manipulations are considered simultaneously).

which the approach ends up producing strategy-proof mechanisms, 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). We emphasize that our goal in this subsection is not necessarily to come up with spectacularly novel mechanisms, but rather to show that the approach advocated in this chapter 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.

Deriving the VCG mechanism

In this subsubsection, 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 from Chapter 4 that the VCG mechanism chooses an outcome that 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 function g.

Because we consider payments part of the outcome in this section, 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 2 *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 we show the main result of this subsubsection, we first characterize optimal manipulations for the agents under the first-price mechanism.

Lemma 22 *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.[6]

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^*.)

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 $\sum\limits_{j \neq i} u_j(\theta_j, s) > u_i(\hat{\theta}_i, s^*) + \sum\limits_{j \neq i} u_j(\theta_j, s^*)$, or equivalently, $VCG_i(\theta_i, \theta_{-i}) < \sum\limits_{j \neq i} u_j(\theta_j, s) - u_j(\theta_j, s^*)$. However, by definition, $VCG_i(\theta_i, \theta_{-i}) = \max_{s^{**}} \sum\limits_{j \neq i} u_j(\theta_j, s^{**}) - u_j(\theta_j, s^*) \geq \sum\limits_{j \neq i} u_j(\theta_j, s) - u_j(\theta_j, s^*)$, 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 $u_i(\hat{\theta}_i, s) + \sum\limits_{j \neq i} u_j(\theta_j, s) \geq \max_{s^{**}} \sum\limits_{j \neq i} u_j(\theta_j, s^{**})$. Using the fact that $VCG_i(\theta_i, \theta_{-i}) = \max_{s^{**}} \sum\limits_{j \neq i} u_j(\theta_j, s^{**}) - u_j(\theta_j, s^*)$, we can rewrite the previous inequality as $u_i(\hat{\theta}_i, s) + \sum\limits_{j \neq i} u_j(\theta_j, s) \geq VCG_i(\theta_i, \theta_{-i}) + \sum\limits_{j \neq i} u_j(\theta_j, s^*)$, or equivalently $u_i(\hat{\theta}_i, s) \geq VCG_i(\theta_i, \theta_{-i}) + \sum\limits_{j \neq i} u_j(\theta_j, s^*) - u_j(\theta_j, s)$. Because $\sum\limits_{j} u_j(\theta_j, s^*) \geq \sum\limits_{j} u_j(\theta_j, s)$, we can rewrite the previous inequality as $u_i(\hat{\theta}_i, s) \geq VCG_i(\theta_i, \theta_{-i}) - u_i(\theta_i, s^*) + u_i(\theta_i, s) + \sum\limits_{j} u_j(\theta_j, s^*) - u_j(\theta_j, s) \geq VCG_i(\theta_i, \theta_{-i}) - u_i(\theta_i, s^*) + u_i(\theta_i, s)$, 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. ■

[6]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.

Theorem 107 *Under the variant of our approach described above, the mechanism resulting after a single iteration is the VCG mechanism.*

Proof: By Observation 2, the naïvely optimal mechanism is the first-price mechanism. When updating the outcome for θ, by Lemma 22, 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, $\sum\limits_i u_i(\theta_i, s)$, minus the sum of the utilities that we must guarantee the agents, that is, $\sum\limits_i u_i(\theta_i, s^*) - VCG_i(\theta_i, \theta_{-i})$. Because $s = s^*$ maximizes $\sum\limits_i u_i(\theta_i, s)$, this means that the most revenue we can hope to extract is $\sum\limits_i VCG_i(\theta_i, \theta_{-i})$, and the VCG mechanism achieves this. ■

Deriving an equal cost sharing mechanism for a nonexcludable public good

We now return to the problem of designing a mechanism for deciding on whether or not a single nonexcludable public good is built. (We studied the automated design of such mechanisms in Chapter 6, Subsection 6.5.2.) Specifically, every agent i has a type v_i (the value of the good to him), and the mechanism decides whether the good is produced, as well as each agent's payment π_i. If the good is produced, we must have $\sum\limits_i \pi_i \geq c$, where c is the cost of producing the good. An individual rationality constraint applies: for each i, $\pi_i \leq \hat{v}_i$ (nobody pays more than his reported value for the good).

In this subsection, rather than use a single variant of our approach, we actually use two distinct phases. Throughout, our objective is to maximize the efficiency of the decision (the good should be produced if and only if the total utility that it generates exceeds the cost of the good); as a secondary objective, we try to maximize revenue.[7] Thus, the naïvely optimal mechanism is to produce the good if and only if the sum of the reported valuations exceeds c, and if so, to charge every agent his entire reported valuation.

An iteration in the first phase proceeds almost exactly as in the previous subsubsection. We try to prevent manipulations according to **(a)** above: for a type vector from which there are beneficial manipulations, make its outcome desirable enough to the manipulating agents to prevent the manipulations, and among outcomes that achieve this, choose the one that maximizes our objective. If there is no such outcome, then we do not change the outcome selected for this type vector. However, in each iteration, we consider only a limited set of manipulations: in the first iteration, we consider only manipulations *to the highest possible type* (*i.e.* the highest possible value that an agent may have for the public good); in the second, we also consider manipulations to the second highest possible type; *etc.*, up until the last iteration, in which we consider manipulations all the way down to a value of 0.

[7]It is not necessary to have this secondary objective for the result to go through, but it simplifies the analysis.

Technically speaking, this approach only works on a finite type space. If the type space is $\mathbb{R}^{\geq 0}$ (all nonnnegative valuations), we encounter two problems: first, there is no highest valuation to start with; and second, there are uncountably infinitely many valuations, leading to infinitely many iterations. Thus, it would not be possible to run the approach automatically in this case. However, for the purpose of theoretical analysis, we can (and will) still consider the case where the type space is $\mathbb{R}^{\geq 0}$: the first problem is overcome by the fact that manipulating to a *higher* type is not beneficial in this domain (free-riders pretend to have a lower valuation); the second problem is overcome by conceiving of this process as the limit of a sequence of similar processes corresponding to finer and finer discretizations of the nonnegative numbers. (If we were to actually run on a discretization, the final resulting mechanism would be close to the mechanism that results in the limit case—and the finer the discretization, the closer the result will be.)

Before we describe the second phase, we will first analyze what happens in the first phase.

Lemma 23 *After considering manipulations to value r, the mechanism will take the following form ($\hat{v}_i$ is agent i's reported value):*

1. *The good will be produced if and only if $\sum_i \hat{v}_i \geq c$;*

2. *If the good is produced, and $\sum_i \min\{r, \hat{v}_i\} > c$, then every agent i will pay $\min\{r, \hat{v}_i\}$;*

3. *If the good is produced, and $\sum_i \min\{r, \hat{v}_i\} \leq c$, then, letting $t \geq r$ be the number such that $\sum_i \min\{t, \hat{v}_i\} = c$, every agent i will pay $\min\{t, \hat{v}_i\}$.*

Proof: Suppose we have proved the result for manipulations to r; let us prove the result for manipulations to an infinitesimally smaller r'. Consider an arbitrary type vector $v = \langle v_1, \ldots, v_n \rangle$ with $\sum_i v_i \geq c$. In the mechanism M that results after considering manipulations to r only, any agent i with $v_i \leq r'$ has no incentive to manipulate to r' (after the manipulation, the agent will be made to pay at least $r' \geq v_i$), so we will not change such an agent's payments. An agent with $v_i > r'$, however, will have an incentive to manipulate to r', *if* this manipulation does not prevent the production of the good (the agent will pay r' rather than the at least $\min\{r, v_i\} > r'$ that he would have paid without manipulation). If the total payment under M given v exceeds c (that is, 2. above applies), then manipulating to r' in fact does not prevent the production of the good, so all such agents have an incentive to manipulate; but, on the other hand, we can reduce the payment of such agents from r to r' in the new mechanism for type vector v, which will prevent the manipulation. However, if the total payment under M given v is exactly c (that is, 3. above applies), then it is impossible to reduce the payments of such agents to r', because we cannot collect any more money from the remaining agents and hence we would not be able to afford the good. ■

Corollary 31 *After considering all manipulations (including to $r = 0$), the mechanism will take the following form:*

1. *The good will be produced if and only if $\sum_i \hat{v}_i \geq c$;*

2. *If the good is produced, then, letting t be the number such that $\sum_i \min\{t, \hat{v}_i\} = c$, every agent i will pay $\min\{t, \hat{v}_i\}$.*

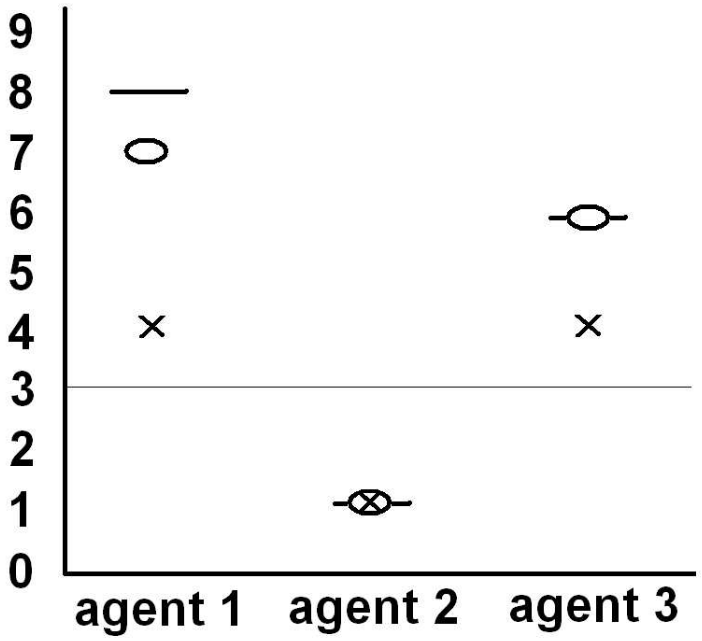

Figure 10.1: Example of a public good setting in which there are 3 agents; the public good costs 9 to produce. The horizontal lines represent the agents' true valuations (8, 1, and 6), which are sufficient to produce the good. The circles represent the payments that the agents make for this type vector after considering manipulations to 7; the crosses represent the payments that the agents make for this type vector after considering manipulations to 4. At this stage the payments sum exactly to 9, so the payments remain at this level even after considering manipulations to even lower values.

The mechanism from Corollary 31 (call it M_1) is still not strategy-proof. For example, in the example in Figure 10.1, suppose that agent 2's valuation for the good is 3 instead. Then, M_1 will charge agent 2 a payment of 3 instead of 1. Thus, agent 2 will be better off reporting 1 instead. However, the next phase will make the mechanism strategy-proof.

In phase two, we take approach **(b)** above: for a type vector *into* which there are beneficial manipulations, make its outcome *un*desirable enough to the manipulating agents to prevent the manipulations. We will do so by not producing the good at all for such type vectors. We will perform a single iteration of this, considering all possible manipulations.

Lemma 24 *A type vector $\langle v_1, \ldots, v_n \rangle$ for which the mechanism M_1 produces the good has a manipulation into it if and only if for some i, $v_i < c/n$.*

Proof: If $v_i < c/n$, consider the modified type vector $\langle v_1, \ldots, v_{i-1}, v_i', v_{i+1}, \ldots, v_n \rangle$ with $v_i' = c/n$. For this modified type vector, i must pay at least c/n (because it must always be the case that

$t \geq c/n$), and thus i would be better off manipulating into the original type vector. This proves the "if" part of the lemma.

On the other hand, if for all i, $v_i \geq c/n$, then $t = c/n$, and all agents pay c/n. Suppose there exists a beneficial manipulation by some agent i into this type vector, from some modified type vector $\langle v_1, \ldots, v_{i-1}, v_i', v_{i+1}, \ldots, v_n \rangle$ for which the good is also produced. (It is never beneficial to manipulate from a type vector for which the good is not produced, as the manipulating agent would have to pay more than his value for the good.) We must have $v_i' > c/n$, for otherwise i would be at least as well off reporting truthfully. But then, everyone pays c/n in the modified type vector as well, contradicting the incentive to manipulate. This proves the "only if" part of the lemma. ■

Thus, we have the following theorem for the mechanism M_2 that results after one iteration of the second phase:

Theorem 108 *M_2 produces the good if and only if for all i, $\hat{v}_i \geq c/n$; and if so, M_2 charges every agent c/n.*

Thus, our approach has produced a very simple mechanism that most of us have encountered at some point in life: the agents are to share the costs of producing the good equally, and if one of them refuses to do so, the good will not be produced (and nobody will have to pay). This mechanism may seem somewhat disappointing, especially if it is unlikely that all the agents will value the good at at least c/n. However, it turns out that *this is in fact the best possible anonymous strategy-proof mechanism* (that satisfies the individual rationality constraint). (Moulin [1994] has already shown a similar result in a more general setting in which multiple levels of the public good can be produced; however, he required *coalitional* strategy-proofness, and he explicitly posed as an open question whether the result would continue to hold for the weaker notion of (individual) strategy-proofness.)

Deriving the plurality-with-runoff rule for voting

In this subsection, we address voting (social choice) settings. Recall that in such a setting, every agent (voter) i's type is a complete ranking $\succ_i$ over the outcomes (candidates). The mechanism (voting rule) takes as input the agents' type reports (votes), consisting of complete rankings of the candidates, and chooses an outcome.

Recall that under the commonly used plurality rule, 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 close to 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. Recall that 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 (Chapter 4), this is still not a strategy-proof mechanism (it is neither dictatorial nor does it preclude any candidate from winning)—indeed, 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 it would either be dictatorial or preclude some candidate from winning).

In this subsubsection, 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 of manipulations that we consider is that 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).

Theorem 109 *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).*[8]

Proof: We will prove the result by induction on $s(b) - s(a)$. First note that there must be some voter that prefers a to b but did not rank a first. Now, if $s(b) - s(a) = 1$, a beneficial manipulation for this voter is to rank a first, which will make a win. No other candidate can be made to win with a beneficial manipulation (no voter ranking b first has an incentive to change his vote, hence no beneficial manipulation will reduce b's score; any other candidate's score can be increased by at most 1 by a manipulation; and every candidate besides a is at least two votes behind b). Thus, in the first iteration, we must decide whether to keep b as the winner, or change it to a. If we keep b as the winner, all the voters that prefer a to b but do not rank a first have an incentive to change their vote (and rank a first). On the other hand, if we change the winner to a, all the voters that prefer b to a but do not rank b first have an incentive to change their vote (and rank b first), so that b leads by two votes and wins. So, in which case do we have more voters with an incentive to change their vote? In the first case, because there are at least as many voters preferring a to b as b to a, and there are fewer voters among those preferring a to b that rank a first than there are voters preferring b to a that rank b first. Hence, we will change the outcome to a.

Now suppose that we have proven the result for $s(b) - s(a) = k - 1$; let us prove it for $s(b) - s(a) = k$. First, we note that in prior iterations, there were no beneficial manipulations from the type vector that we are considering (no voter ranking b first has an incentive to change his vote, thus any beneficial manipulation can only reduce the difference between $s(b)$ and the score of another candidate by 1, and by the induction assumption no vote vector that results from such a manipulation has had its outcome changed in earlier iterations—*i.e.* it is still b), and thus the outcome has not yet changed in prior iterations. But, by the induction assumption, any vote vector

[8]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.)

that can be obtained from the vote vector that we are considering by changing one of the votes ranking a higher than b but not first, to one that ranks a first, must have had its outcome changed to a in the previous round. Thus, now there is a beneficial manipulation that will make a the winner. On the other hand, no other candidate can be made to win by such a beneficial manipulation, since they are too far behind b given the current number of iterations. The remainder of the analysis is similar to the base case ($s(b) - s(a) = 1$). ■

10.1.4 Computing the outcomes of the mechanism

In this subsection, 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. One potential upside relative to standard automated mechanism design techniques (as presented in Chapter 6) is that we do not need to compute the entire mechanism (the outcomes for all type vectors) here; 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 manipulations). 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.

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.

10.1.5 Computational hardness of manipulation

We have already demonstrated that our approach can change naïve mechanisms into mechanisms that are less (sometimes not at all) manipulable. In this subsection, 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 computationally too difficult to discover beneficial manipulations, the revelation principle ceases to meaningfully apply, and a manipulable mechanism can sometimes actually outperform all truthful mechanisms.

In this subsection, we first present 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. This argument depends on the assumption that the agents only use the most straightforward possible algorithm for finding manipulations. Second, we show that if we add a random component to our approach for updating the mechanism, then we can prove formally that detecting whether there is a beneficial manipulation becomes #P-hard.

An informal argument for hardness of manipulation

Suppose that the only thing that an agent knows 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, one 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 is computationally infeasible if the agent does not have the computational capabilities to simulate as many iterations as the designer will actually perform.

Of course, this argument fails if the agent actually has greater computational abilities or better algorithms than the designer. In the next subsubsection, we will give a different, formal argument for hardness of manipulation for one particular instantiation of our approach.

Random sequential updating leads to #P-hardness

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).

The hardness result that we show in this subsubsection holds even for a single agent; therefore we will only specify the variant of our approach used in this subsubsection for a single agent. Any generalization of the variant to more than one agent will have the same property.

First, we take the set Θ of all of the agent's types, and organize the types as a sequence of $|\Theta|/2$ pairs of types:[9] $((\theta_{11}, \theta_{12}), (\theta_{21}, \theta_{22}), \ldots, (\theta_{|\Theta|1}, \theta_{|\Theta|2}))$. In the ith iteration, we randomly choose between θ_{i1} and θ_{i2}, and consider all the beneficial manipulations out of the chosen type (and no other manipulations). (We will only need $|\Theta|/2$ iterations.) Then, as before, we try to prevent these manipulations by making the outcome for the chosen type more appealing to an agent with that type (and if there are multiple ways of doing so, we choose the one that maximizes the designer's objective).

We are now ready to present our #P-hardness result. We emphasize that, similarly to the hardness results in Chapter 8, this is only a worst-case notion of hardness, which may not prevent manipulation in all cases.

Theorem 110 *Evaluating whether there exists a manipulation that increases an agent's expected utility is #P-hard*[10] *under the variant of our technique described above.*

Proof: We reduce an arbitrary #SAT instance, together with a number K, to the following setting. Let there be a single agent with the following types. For every variable $v \in V$, we have types θ_{+v} and θ_{-v}; for every clause $c \in C$, we have types θ_c^1 and θ_c^2; finally, we have four additional types $\theta_1, \theta_2, \theta_3, \theta_4$. The sequence of pairs of types is as follows: $((\theta_{+v_1}, \theta_{-v_1}), \ldots, (\theta_{+v_{|V|}}, \theta_{-v_{|V|}}),$ $(\theta_{c_1}^1, \theta_{c_1}^2), \ldots, (\theta_{c_{|C|}}^1, \theta_{c_{|C|}}^2), (\theta_1, \theta_2), (\theta_3, \theta_4))$. Let the outcome set be as follows: for every variable $v \in V$, we have outcomes o_{+v} and o_{-v}; for every clause $c \in C$, we have an outcome o_c; finally, we have outcomes o_1, o_2, o_3, o_4. The utility function is zero everywhere, with the following exceptions: for every literal l, $u(\theta_l, o_l) = 2, u(\theta_l, o_2) = 1$; for every clause c, $u(\theta_c, o_c) = 4, u(\theta_c, o_l) = 3$ if l occurs in c, $u(\theta_c, o_3) = 2, u(\theta_c, o_2) = 1$; for $\theta \in \{\theta_1, \theta_2\}$, $u(\theta, o_4) = \frac{2^{|V|+1}}{2^{|V|}-K}, u(\theta, o_1) = 1$; for $\theta \in \{\theta_3, \theta_4\}$, $u(\theta, o_4) = 2, u(\theta, o_3) = 1$. The designer's objective function is zero everywhere, with the following exceptions: for all $\theta \in \Theta$, $g(\theta, o_1) = 3$; $g(\theta_3, o_2) = g(\theta_4, o_2) = 4$; for every literal l, $g(\theta_l, o_l) = 2$; for every clause c, $g(\theta_c, o_3) = 2, g(\theta_c, o_c) = 1$; $g(\theta_3, o_4) = g(\theta_4, o_4) = 2$. The initial mechanism, which naïvely maximizes the designer's objective, chooses o_1 for all types, with the exception of θ_3 and θ_4, for which it chooses o_2.

The mechanism will first update exactly one of θ_{+v} and θ_{-v}, for every $v \in V$. Specifically, if θ_l (where l is a literal) is updated, the new outcome chosen for that type will be o_l (which is more desirable to the agent that o_2, and better for the designer than o_2). Subsequently, exactly one of θ_c^1 and θ_c^2 is updated. Specifically, if θ_c^i is updated, the new outcome chosen for that type will be o_3 if no type θ_l with $l \in c$ has been updated (and therefore no o_l with $l \in c$ is ever chosen by the mechanism), and o_c otherwise. (o_3 is more desirable to the agent than o_2, but less desirable than some o_l with $l \in c$; however, o_c is even more desirable than such an o_l. The designer would prefer to prevent the manipulation with o_3, but o_c is the next best way of preventing the manipulation if o_3 will not suffice.) Then, one of θ_1 and θ_2 is updated, but the outcome will not be changed for either of them (the only outcome that the agent would prefer to o_1 for these types is o_4, which the mechanism does not yet choose for any type); finally, one of θ_3 and θ_4 is updated, and the outcome for this

[9]The hardness result will hold even if the number of types is restricted to be even, so it does not matter how this is generalized to situations in which the number of types is odd.

[10]Technically, we reduce from a decision variant of #SAT ("Are there fewer than K solutions?"). An algorithm for this decision variant can be used to solve the original problem using binary search.

type will change to o_4 if and only if outcome o_3 is now chosen by the mechanism for some type θ_c^i (that outcome would be preferred to o_2 by the agent for these types; o_4 is the next best outcome for the designer). We note that this update to o_4 will *not* happen if and only if, for every clause c, for at least one literal $l \in c$, θ_l was updated (rather than θ_{-l})—because in this case (and only in this case), whenever we updated one of θ_c^1, θ_c^2, o_c was chosen (rather than o_3). In other words, it will not happen if and only if the literals l that were chosen constitute a satisfying assignment for the formula. The probability that this happens is $n/(2^{|V|})$, where n is the number of solutions to the SAT formula.

Now, let us consider whether the agent has an incentive to manipulate if his type is θ_1. Reporting truthfully will lead to outcome o_1 being chosen, giving the agent a utility of 1. It only makes sense to manipulate to a type for which o_4 may be chosen, because o_1 is preferred to all other outcomes—hence, any beneficial manipulation would be to θ_3 or θ_4. Without loss of generality, let us consider a manipulation to θ_3. What is the chance (given that we have done exactly $|\Theta|/2$ updates) that we choose o_4 for θ_3? It is $1/2$ (the chance that θ_3 was in fact updated) times $1 - n/(2^{|V|})$ (the chance that o_4 was chosen), which is $(2^{|V|} - n)/(2^{|V|+1})$. Otherwise, o_2 is still chosen for θ_3, and manipulating to θ_3 from θ_1 would give utility 0. Thus, the expected utility of the manipulation is $\frac{2^{|V|}-n}{2^{|V|+1}} \frac{2^{|V|+1}}{2^{|V|}-K}$. This is greater than 1 if and only if $n < K$. ■

10.2 Summary

In this chapter, we suggested an approach for (automatically) designing mechanisms for bounded agents. Under this approach, we start with a naïve (manipulable) mechanism, and incrementally make it *more* strategy-proof over a sequence of iterations.

We gave various examples of mechanisms that (variants of) our approach generate, including: the VCG mechanism in general settings with payments; an equal cost sharing mechanism for public goods settings; and the plurality-with-runoff voting rule. We also provided several basic algorithms for automatically executing our approach in general settings, including a recursive algorithm and a dynamic programming algorithm, and analyzed their running times. Finally, we discussed how computationally hard it is for agents to find any remaining beneficial manipulation. We argued that agents using a straightforward manipulation algorithm will not be able to compute manipulations if the designer has greater computational power (and can thus execute more iterations). We also showed that if we add randomness to how the outcomes are computed, then detecting whether a manipulation that is beneficial in expectation exists becomes #P-hard for an agent.