Chapter 6

Automated Mechanism Design

Mechanism design has traditionally been a manual endeavor. The designer uses experience and intuition to hypothesize that a certain rule set is desirable in some ways, and then tries to prove that this is the case. Alternatively, the designer formulates the mechanism design problem mathematically and characterizes desirable mechanisms analytically in that framework. These approaches have yielded a small number of canonical mechanisms over the last 40 years, the most significant of which we discussed in Chapter 4. Each of these mechanisms is designed for a class of settings and a specific objective. The upside of these mechanisms is that they do not rely on (even probabilistic) information about the agents' preferences (*e.g.* Vickrey-Clarke-Groves mechanisms), or they can be easily applied to any probability distribution over the preferences (*e.g.* the dAGVA mechanism, the Myerson auction, and the Maskin-Riley multi-unit auction). However, these general mechanisms also have significant downsides:

- The most famous and most broadly applicable general mechanisms, VCG and dAGVA, only maximize social welfare. If the designer is self-interested, as is the case in many electronic commerce settings, these mechanisms do not maximize the designer's objective.

- The general mechanisms that do focus on a self-interested designer are only applicable in very restricted settings. For example, Myerson's expected revenue maximizing auction is for selling a single item, and Maskin and Riley's expected revenue maximizing auction is for selling multiple identical units of an item.

- Even in the restricted settings in which these mechanisms apply, the mechanisms only allow for payment maximization. In practice, the designer may also be interested in the outcome *per se*. For example, an auctioneer may care which bidder receives the item.

- It is often assumed that side payments can be used to tailor the agents' incentives, but this is not always practical. For example, in barter-based electronic marketplaces—such as mybarterclub.com, Recipco, and National Trade Banc—side payments are not allowed. Furthermore, among software agents, it might be more desirable to construct mechanisms that do not rely on the ability to make payments, because many software agents do not have the infrastructure to make payments.

- The most common mechanisms (*e.g.*, VCG, dAGVA, the Myerson auction, and the Maskin-Riley auction) assume that the agents have quasilinear preferences—that is, they assume that the utility function of each agent $i \in \{1, \ldots, n\}$ can be written as $u_i(o, \pi_1, \ldots, \pi_n) = v_i(o) - \pi_i$, where o is the outcome and π_i is the amount that agent i has to pay. So, very restrictively, it is assumed that 1) the agent's valuation, v_i, of outcomes is independent of money, 2) the agent does not care about other agents' payments, and 3) the agent is risk neutral.

In sharp contrast to manual mechanism design, in this chapter we introduce a systematic approach—called *automated mechanism design (AMD)*—where the mechanism is automatically created for the setting and objective at hand [Conitzer and Sandholm, 2002b].[1] This has at least four important advantages:

- It can be used in settings beyond the classes of problems that have been successfully studied in (manual) mechanism design to date.
- It can allow one to circumvent the impossibility results: when the mechanism is designed for the setting (instance) at hand, it does not matter that it would not work on preferences beyond those in that setting (*e.g.*, for a class of settings). Even when the optimal mechanism—created automatically—does not circumvent the impossibility, it always minimizes the pain entailed by impossibility.
- It can yield better mechanisms (in terms of better outcomes and/or stronger nonmanipulability guarantees[2]) than the canonical mechanisms because the mechanism capitalizes on the particulars of the setting (the probabilistic (or other) information that the mechanism designer has about the agents' preferences). Given the vast amount of information that parties have about each other today, it is astonishing that the canonical mechanisms (such as first-price reverse auctions), which ignore that information, have prevailed thus far. It seems likely that future mechanisms will be created automatically. For example, imagine a Fortune 1000 company automatically creating its procurement mechanism based on its statistical knowledge about its suppliers (and potentially also the public prices of the suppliers' inputs, *etc.*). Initial work like this is already being conducted at CombineNet, Inc.
- It shifts the burden of mechanism design from humans to a machine.

The rest of this chapter is layed out as follows. In Section 6.1, we define the basic computational problem of automated mechanism design [Conitzer and Sandholm, 2002b]. In Section 6.2, we illustrate the types of mechanism that automated mechanism design can create, using divorce settlement as an example [Conitzer and Sandholm, 2003a]. In Section 6.3, we show that several variants of the problem of designing an optimal deterministic mechanism are hard [Conitzer and Sandholm, 2003b,

[1] Automated mechanism design is completely different from *algorithmic mechanism design* [Nisan and Ronen, 2001]. In the latter, the mechanism is designed manually with the goal that *executing* the mechanism is computationally tractable. On the other hand, in automated mechanism design, the mechanism itself is designed automatically. Some work on automatically choosing the mechanism to use that preceded our work was done by Cliff [2001], Byde [2003], and Phelps *et al.* [2002]. These works focused on setting a parameter of the mechanism (rather than searching through the space of all possible mechanisms, as we do here), and evaluated the resulting mechanism based on agents that they evolved with the mechanism (rather than requiring truthfulness).

[2] For example, satisfaction of *ex post* IC and/or IR constraints rather than their *ex interim* variants.

2004f]. In Section 6.4, we show that optimal randomized mechanisms can be designed in polynomial time using a linear programming formulation, and that a mixed integer programming version of this formulation can be used to design optimal deterministic mechanisms [Conitzer and Sandholm, 2002b, 2003b, 2004f]. In Section 6.5, we demonstrate some initial applications of automated mechanism design [Conitzer and Sandholm, 2003a]. In Section 6.6, we give experimental scalability results for the linear/mixed integer programming techniques Conitzer and Sandholm [2003a]. In Section 6.7, we give and study a special-purpose algorithm for the special case of designing a deterministic mechanism for a single agent that does not use payments [Conitzer and Sandholm, 2004a]. In Section 6.8, we introduce a representation that can be more concise than the straightforward representation of automated mechanism design problem instances, and study how using this representation affects the complexity of the problem [Conitzer and Sandholm, 2003c].

6.1 The computational problem

In this section, we define the computational problem of automated mechanism design. First, we define an instance of the problem as follows.

Definition 24 *In an* automated mechanism design setting*, we are given*
A finite set of outcomes O;
A finite set of n agents;
For each agent i,

- *a finite*[3] *set of* types *set of types Θ_i,*
- *a probability distribution γ_i over Θ_i (in the case of correlated types, there is a single joint distribution γ over $\Theta_1 \times \ldots \times \Theta_n$),*
- *a utility function $u_i : \Theta_i \times O \rightarrow \mathbb{R}$;*

An objective function whose expectation the designer wishes to maximize.

There are many possible objective functions the designer might have, for example, social welfare (where the designer seeks to maximize the sum of the agents' utilities, $\sum_{i=1}^{n} u_i(\theta, o)$, or $\sum_{i=1}^{n} u_i(\theta, o) - \pi_i$ if payments are taken into account), or the minimum utility of any agent (where the designer seeks to maximize the worst utility had by any agent, $\min_i u_i(\theta, o)$, or $\min_i u_i(\theta, o) - \pi_i$ if payments are taken into account). In both of these cases, the designer is *benevolent*, because the designer, in some sense, is pursuing the agents' collective happiness. On the other hand, a *self-interested* designer cares only about the outcome chosen (that is, the designer does not care how the outcome relates to the agents' preferences, but rather has a fixed preference over the outcomes), and about the net payments made by the agents, which flow to the designer. Specifically, a self-interested designer

[3] It should be noted that in mechanism design, the type space is often continuous. However, the techniques described in this chapter require a finite number of types. One can approximate a continuous type space with a discretized type space, but perhaps future research will discover more elegant and better methods for dealing with continuous type spaces.

has an objective function $g(o) + \sum_{i=1}^{n} \pi_i$, where $g : O \rightarrow \mathbb{R}$ indicates the designer's own preference over the outcomes, and π_i is the payment made by agent i. In the case where $g = 0$ everywhere, the designer is said to be *payment maximizing*. In the case where payments are not possible, g constitutes the objective function by itself.

We can now define the computational problem of automated mechanism design.

Definition 25 *(AUTOMATED-MECHANISM-DESIGN (AMD)) We are given an automated mechanism design setting, an IR notion (*ex interim, ex post, *or none), and a solution concept (dominant strategies or Bayes-Nash equilibrium). Also, we are told whether payments are possible, and whether randomization is possible. Finally, we are given a target value G. We are asked whether there exists a mechanism of the specified type that satisfies both the IR notion and the solution concept, and gives an expected value of at least G for the objective.*[4]

6.2 A tiny example: Divorce settlement

To get some intuition about the types of mechanism that AMD generates, in this section, we apply AMD to divorce settlement. We study several variants of the mechanism design problem, and the optimal solutions (mechanisms) to those variants generated by our AMD implementation (described later). We first study a benevolent arbitrator, then a benevolent arbitrator that uses payments to structure the agents' incentives, and finally a greedy arbitrator that wants to maximize the sum of side payments from the agents—while still motivating the agents to come to the arbitration.

6.2.1 A benevolent arbitrator

A couple is getting a divorce. They jointly own a painting and the arbitrator has to decide what happens to the painting. There are 4 options to decide among: (1) the husband gets the painting, (2) the wife gets the painting, (3) the painting remains in joint ownership and is hung in a museum, and (4) the painting is burned. The husband and wife each have two possible types: one that implies not caring for the painting too much (low), and one that implies being strongly attached to the painting (high). (low) is had with probability .8, (high) with .2, by each party. To maximize social welfare, the arbitrator would like to give the painting to whoever cares for it more, but even someone who does not care much for it would prefer having it over not having it, making the arbitrator's job in ascertaining the preferences nontrivial. Specifically, the utility function is (for either party)

```
u(low,get the painting)=2
u(low,other gets the painting)=0
u(low,joint ownership)=1
u(low,burn the painting)=-10 (both parties feel
 that burning the painting would be a terrible
 thing from an art history perspective)
u(high,get the painting)=100
```

[4]For studying computational complexity, we phrase AMD as a decision problem, but the corresponding optimization problem is clear.

```
u(high,other gets the painting)=0
u(high,joint ownership)=50
u(high,burn the painting)=-10
```

Let us assume (for now) that side payments are not possible, randomization is not possible, and that implementation in dominant strategies is required. Now we have a well-specified AMD instance. Our solver generated the following optimal mechanism for this setting:

```
           husband_low            husband_high
wife_low   husband gets painting  husband gets painting
wife_high  husband gets painting  husband gets painting
```

That is, we cannot do better than always giving the painting to the husband (or always giving it to the wife). (The solver does not look for the "fairest" mechanism because fairness is not part of the objective we specified.) Now let us change the problem slightly, by requiring only implementation in BNE. For this instance, our solver generated the following optimal mechanism:

```
           husband_low         husband_high
wife_low   joint ownership     husband gets painting
wife_high  wife gets painting  painting is burned
```

Thus, when we relax the incentive compatibility constraint to BNE, we can do better by sometimes burning the painting! The burning of the painting (with which nobody is happy) is sufficiently helpful in tailoring the incentives that it becomes a key part of the mechanism. (This is somewhat similar to the item not being sold in an optimal (*i.e.*, revenue-maximizing) auction—more on optimal auctions later.) Now let us see whether we can do better by also allowing for randomization in the mechanism. It turns out that we can, and the optimal mechanism generated by the solver is the following:

```
           husband_low              husband_high
wife_low   .57: husband, .43: wife  1: husband
wife_high  1: wife                  .45: burn; .55: husband
```

The randomization helps us because the threat of burning the painting *with some probability* when both report high is enough to obtain the incentive effect that allows us to give the painting to the right party in other settings. Interestingly, the mechanism now chooses to randomize over the party that receives the painting rather than awarding joint ownership in the setting where both report low.

6.2.2 A benevolent arbitrator that uses payments

Now imagine that we can force the parties to pay money, depending on the types reported—that is, side payments are possible. The arbitrator (for now) is still only concerned with the parties' welfare—taking into account how much money they lose because of the payment rule, as well as the

allocation of the painting.[5] Thus, it does not matter to the arbitrator whether the agents' net payment goes to the arbitrator, a charity, or is burned, but other things being equal the arbitrator would like to minimize the payments that the agents make. Now the optimal deterministic mechanism in dominant strategies generated by the solver has the following allocation rule:

```
           husband_low            husband_high
wife_low   husband gets painting  husband gets painting
wife_high  wife gets painting     wife gets painting
```

The payment function is (wife's payment listed first):

```
           husband_a  husband_high
wife_low   0,0        0,0
wife_high  2,0        2,0
```

In this mechanism, the allocation of the painting is always optimal. However, the price (in terms of social welfare) that is paid for this is that the wife must sometimes pay money; the fact that she has to pay 2 whenever she reports her high type removes her incentive to falsely report her high type.

6.2.3 An arbitrator that attempts to maximize the payments extracted

Now we imagine a non-benevolent arbitrator, who is running an arbitration business. The agents' net payments now go to the arbitrator, who is seeking to maximize these payments. Of course, the arbitrator cannot extract arbitrary amounts from the parties; rather, the parties should overall still be happy with their decision to go to the arbitrator. Thus, we need an IR constraint. If we require ex post IR and dominant strategies, the optimal deterministic mechanism generated by the solver has the following allocation rule:

```
           husband_low          husband_high
wife_low   painting is burned   husband gets painting
wife_high  wife gets painting   wife gets painting
```

Now the painting is burned when both parties report their low types! (This is even more similar to an item not being sold in an optimal combinatorial auction.) As for the mechanism's payment function: in this setting, the arbitrator is always able to extract *all* of each agent's utility from the allocation as her payment (but note that the allocation is not always optimal: the painting is burned sometimes, in which case the arbitrator obtains no revenue, but rather has to compensate the parties involved for the loss of the painting).

Many other specifications of the problem are possible, but we will not study them here.

[5] Classical mechanism design often does not count the payments in the social welfare calculation (*e.g.*, the VCG mechanism), allowing for easier analysis; one of the benefits of automated mechanism design is that the payments made can easily be integrated into the social welfare calculation in designing the mechanisms.

6.3 Complexity of designing deterministic mechanisms

This section characterizes the computational complexity of automated mechanism design for the case where the designed mechanism is required to be deterministic. An interesting special case is the setting where there is only one agent (or, more generally, only one *type-reporting* agent). In this case, the agent always knows everything there is to know about the other agents' types—because there is nothing to know about their types. Since *ex post* and *ex interim* IR only differ on what an agent is assumed to know about other agents' types, the two IR concepts coincide here. Also, because implementation in dominant strategies and implementation in Bayes-Nash equilibrium only differ on what an agent is assumed to know about other agents' types, the two solution concepts coincide here. This observation is a useful tool in proving hardness results: if we prove computational hardness in the single-agent setting, this immediately implies hardness for both IR concepts, for both solution concepts, and for any constant number of agents.

In this section, we will show that most variants of the automated mechanism design problem are hard (NP-complete) even in the single-agent setting, *if* the mechanism is required to be deterministic. (In contrast, we will show in Section 6.4 that allowing for randomized mechanisms makes the problem solvable in polynomial time.) Most of the reductions are from the MINSAT problem:

Definition 26 (MINSAT) *We are given a formula ϕ in conjunctive normal form, represented by a set of Boolean variables V and a set of clauses C, and an integer K ($K < |C|$). We are asked whether there exists an assignment to the variables in V such that at most K clauses in ϕ are satisfied.*

MINSAT was recently shown to be NP-complete [Kohli *et al.*, 1994].

We first show that the problem of designing a welfare-maximizing deterministic mechanism that does not use payments is NP-complete. Of course, this problem is easy if there is only a single agent: in this case, the welfare-maximizing mechanism is to always give the agent one of its most preferred outcomes. However, we show that if, in addition to the type-reporting agent, there is an additional agent that does not report a type (for example, because its type is common knowledge), then the problem becomes NP-complete.

Theorem 37 *The AMD problem for designing deterministic mechanisms without payments is NP-complete, even when the objective is social welfare, there is only a single type-reporting agent (in addition to an agent that does not report a type), and the probability distribution over Θ is uniform. (Membership in NP is guaranteed only if the number of agents is constant.)*

Proof: The problem is in NP when the number of agents is constant because we can nondeterministically generate an outcome selection function, and subsequently verify in polynomial time whether it is nonmanipulable, and whether the expectation of the objective function achieves the threshold. (We note that if we do not restrict the number of agents, then the outcome selection function will have exponential size.) To show that the problem is NP-hard, we reduce an arbitrary MINSAT instance to an automated mechanism design instance as follows.

Let the outcomes O be as follows. For every clause $c \in C$, there is an outcome o_c. For every variable $v \in V$, there is an outcome o_v and an outcome o_{-v}. Finally, there is a single additional outcome o_b.

Let L be the set of literals, that is, $L = \{v : v \in V\} \cup \{-v : v \in V\}$. Then, let the type space Θ be as follows. For every clause $c \in C$, there is a type θ_c. For every variable $v \in V$, there is a type θ_v. The probability distribution over Θ is uniform.

Let the utility function be as follows:

- $u(\theta_v, o_v) = u(\theta_v, o_{-v}) = |C| + 3$ for all $v \in V$;
- $u(\theta_c, o_l) = 1$ for all $c \in C$ and $l \in c$ (that is, l is a literal that occurs in c);
- $u(\theta_c, o_c) = 1$ for all $c \in C$;
- u is 0 everywhere else.

Let $g(\theta, o) = u(\theta, o) + v(o)$, where $v(o_b) = 2$ and v is 0 everywhere else. (Here, v represents the utility of the agent that does not report a type.) Finally, let $G = \frac{|V|(|C|+3)+2|C|-k}{|V|+|C|}$ (k is the threshold of the MINSAT instance). We claim that the automated mechanism design instance has a solution if and only if the MINSAT instance has a solution.

First suppose that the MINSAT instance has a solution, that is, an assignment to the variables that satisfies at most k clauses. Then consider the following mechanism. If $v \in V$ is set to *true* in the assignment, then set $o(\theta_v) = o_v$; if it is set to *false*, then set $o(\theta_v) = o_{-v}$. If $c \in C$ is satisfied by the assignment, then set $o(\theta_c) = o_c$; if it is not satisfied, then set $o(\theta_c) = o_b$. First we show that this mechanism is nonmanipulable. If the agent's type is either any one of the θ_v or one of the θ_c corresponding to a satisfied clause c, then the mechanism gives the agent the maximum utility it can possibly get with that type, so there is no incentive for the agent to misreport. On the other hand, if the agent's type is one of the θ_c corresponding to a nonsatisfied clause c, then any outcome o_l corresponding to a literal l in c, or o_c, would give utility 1, as opposed to o_b (which the mechanism actually chooses for θ_c) which gives the agent utility 0. It follows that the mechanism is nonmanipulable if and only if there is no other θ such that $o(\theta)$ is any outcome o_l corresponding to a literal l in c, or o_c. It is easy to see that there is indeed no θ such that $o(\theta) = o_c$. There is also no θ such that $o(\theta)$ is any outcome o_l corresponding to a literal l in c: this is because the only type that could possibly give the outcome o_l is θ_v, where v is the variable corresponding to l; but because c is not satisfied in the assignment to the variables, we know that actually, $o(\theta_v) = o_{-l}$ (that is, the outcome corresponding to the opposite literal is chosen). It follows that the mechanism is indeed nonmanipulable. All that is left to show is that the expected value of $g(\theta, o(\theta))$ reaches G. For any θ_v we have $g(\theta_v, o(\theta_v)) = |C| + 3$. For any θ_c where c is a satisfied clause, we have $g(\theta_c, o(\theta_c)) = 1$. Finally, for any θ_c where c is an unsatisfied clause, we have $g(\theta_c, o(\theta_c)) = 2$. If s is the number of satisfied clauses, then, using the facts that the probability distribution over Θ is uniform and that $s \leq k$, we have $E[g(\theta, o(\theta))] = \frac{|V|(|C|+3)+s+2(|C|-s)}{|V|+|C|} \geq \frac{|V|(|C|+3)+2|C|-k}{|V|+|C|} = G$. So there is a solution to the automated mechanism design instance.

Now suppose there is a solution to the automated mechanism design instance, that is, a nonmanipulable mechanism given by an outcome function $o : \Theta \to O$, which leads to an expected value of $g(\theta, o(\theta))$ of at least G. We observe that the maximum value that we can get for $g(\theta, o(\theta))$ is $|C| + 3$ when θ is one of the θ_v, and 2 otherwise. Thus, if for some v it were the case that $o(\theta_v) \notin \{o_v, o_{-v}\}$ and hence $g(\theta, o(\theta)) \leq 2$, it would follow that $E[g(\theta, o(\theta))]$ can be at most

$\frac{(|V|-1)(|C|+3)+2(|C|+1)}{|V|+|C|} < \frac{(|V|)(|C|+3)+|C|}{|V|+|C|} < \frac{|V|(|C|+3)+2|C|-k}{|V|+|C|} = G$ (because $k < |C|$). (Contradiction.) It follows that for all v, $o(\theta_v) \in \{o_v, o_{-v}\}$. From this we can derive an assignment to the variables: set v to *true* if $o(\theta_v) = o_v$, and to *false* if $o(\theta_v) = o_{-v}$. We claim this assignment is a solution to the MINSAT instance for the following reason. If a clause c is satisfied by this assignment, there is some literal l such that $l \in c$ and $o(\theta_v) = o_l$ for the corresponding variable v. But then $o(\theta_c)$ cannot be o_b, because if it were, the agent would be motivated to report θ_v when its true type is θ_c, to get a utility of 1 as opposed to the 0 it would get for reporting truthfully. Hence $g(\theta_c, o(\theta_c))$ can be at most 1 for a satisfied clause c. It follows that $E[g(\theta, o(\theta))]$ can be at most $\frac{|V|(|C|+3)+s+2(|C|-s)}{|V|+|C|}$ where s is the number of satisfied clauses. But because $E[g(\theta, o(\theta))] \geq G$, we can conclude $\frac{|V|(|C|+3)+s+2(|C|-s)}{|V|+|C|} \geq G = \frac{|V|(|C|+3)+2|C|-k}{|V|+|C|}$, which is equivalent to $s \leq k$. So there is a solution to the MINSAT instance. ■

We note that the previous result is in contrast to the case where payments are allowed: in that case, the VCG mechanism constitutes an optimal mechanism, and it can be computed in polynomial time. We may wonder if the ability to use payments makes the automated mechanism design problem easy in all cases. The following theorem shows that this is not the case: there are objective functions (that do not depend on the payments made) such that designing the optimal deterministic mechanism is hard even when the mechanism is allowed to use payments.

Theorem 38 *The AMD problem for designing deterministic mechanisms with payments is NP-complete, even when the objective does not depend on the payments made, there is only a single agent, and the probability distribution over Θ is uniform. (Membership in NP is guaranteed only if the number of agents is constant.)*

Proof: First we show that the problem is in NP. When the number of agents is constant, we can nondeterministically generate an outcome function o. We then check whether the payment function π can be set so as to make the mechanism nonmanipulable. Because we have already generated o, we can phrase this problem as a linear program with the following constraints: for all $\theta, \hat{\theta} \in \Theta$, $u(\theta, o(\theta)) + \pi(\theta) \geq u(\theta, o(\hat{\theta})) + \pi(\hat{\theta})$. If the linear program has a solution, we subsequently check if the corresponding mechanism achieves the threshold G for $E[g(\theta, o(\theta))]$.

To show that the problem is NP-hard, we reduce an arbitrary INDEPENDENT-SET instance to an automated mechanism design instance as follows. For every vertex $v \in V$, let there be outcomes o_v^1 and o_v^2, and a type θ_v. The probability distribution over Θ is uniform. Let the utility function be as follows:

- $u(\theta_v, o_w^1) = 1$ for all $v, w \in V$ with $(v, w) \in E$;
- $u(\theta_v, o_w^1) = 0$ for all $v, w \in V$ with $(v, w) \notin E$ (this includes all cases where $v = w$ as there are no self-loops in the graph);
- $u(\theta_v, o_v^2) = 1$ for all $v \in V$;
- $u(\theta_v, o_w^2) = 0$ for all $w \in V$ with $v \neq w$.

Let the objective function be $g(\theta_v, o_v^1) = 1$ for all $v \in V$, and $g() = 0$ everywhere else. Finally, let $G = \frac{k}{|V|}$ (where k is the threshold of the INDEPENDENT-SET instance). We claim that the automated mechanism design instance has a solution if and only if the INDEPENDENT-SET instance has a solution.

First suppose that the INDEPENDENT-SET instance has a solution, that is, some $I \subseteq V$ of size at least k such that no two elements of I have an edge between them. Then consider the following mechanism. For all $v \in I$, let $o(\theta_v) = o_v^1$. For all $v \notin V$, let $o(\theta_v) = o_v^2$. Let π be zero everywhere (no payments are made). First we show that this mechanism is indeed nonmanipulable. If $v \in I$ and $w \in I$, then (because I is an independent set) $(v, w) \notin I$, and thus $u(\theta_v, o(\theta_v)) + \pi(\theta_v) = u(\theta_v, o_v^1) = 0 = u(\theta_v, o_w^1) = u(\theta_v, o(\theta_w)) + \pi(\theta_w)$. If $v \in I$ and $w \notin I$, then $u(\theta_v, o(\theta_v)) + \pi(\theta_v) = u(\theta_v, o_v^1) = 0 = u(\theta_v, o_w^2) = u(\theta_v, o(\theta_w)) + \pi(\theta_w)$. Finally, if $v \notin I$, then $u(\theta_v, o(\theta_v)) + \pi(\theta_v) = u(\theta_v, o_v^2) = 1$, which is the highest possible value the agent can attain. So there is no incentive for the agent to misreport anywhere. All that is left to show is that the expected value of $g(\theta, o(\theta))$ reaches G. For $v \in I$, $g(\theta, o(\theta)) = g(\theta, o_v^1) = 1$, and for $v \notin I$, $g(\theta, o(\theta)) = g(\theta, o_v^2) = 0$. Because the distribution over Θ is uniform, it follows that $E[g(\theta, o(\theta))] = \frac{|I|}{|V|} \geq \frac{k}{|V|} = G$. So there is a solution to the automated mechanism design instance.

Now suppose there is a solution to the automated mechanism design instance, that is, a nonmanipulable mechanism given by an outcome function $o : \Theta \to O$ and a payment function $\pi : \Theta \to \mathbb{R}$, which leads to an expected value of $g(\theta, o(\theta))$ of at least G. Let $I = \{v : o(\theta) = o_v^1\}$. We claim I is a solution to the INDEPENDENT-SET instance. First, because $g(\theta_v, o(\theta_v))$ is 1 only for $v \in I$, we know that $\frac{k}{|V|} = G \leq E[g(\theta, o(\theta))] = \frac{|I|}{|V|}$, or equivalently, $|I| \geq k$. All that is left to show is that there are no edges between elements of I. Suppose there were an edge between $v, w \in I$. Without loss of generality, say $\pi(\theta_v) \leq \pi(\theta_w)$. Then, $u(\theta_v, o(\theta_v)) + \pi(\theta_v) = u(\theta_v, o_v^1) + \pi(\theta_v) = \pi(\theta_v) \leq \pi(\theta_w) < 1 + \pi(\theta_w) = u(\theta_v, o_w^1) + \pi(\theta_w) = u(\theta_v, o(\theta_w)) + \pi(\theta_w)$. So the agent has an incentive to misreport when its type is θ_v, which contradicts the nonmanipulability of the mechanism. It follows that there are no edges between elements of I. So there is a solution to the INDEPENDENT-SET instance. ■

The objective functions studied up to this point depended on the agents' types. However, this is not the case for so-called *self-interested* designers, who are concerned only with how the chosen outcome fits their own goals and the payments collected. Formally, we say that the designer is self-interested if the objective function takes the form $g(o) + \sum_{i=1}^{n} \pi_i$, where $g : O \to \mathbb{R}$ indicates the designer's own preference over the outcomes, and π_i is the payment made by agent i. While the previous complexity results did not depend on the presence of an IR constraint, the automated mechanism design problem is trivial for a self-interested designer without an IR constraint: the designer can simply choose the outcome that it likes best, and force the agents to pay an unbounded amount. Hence, the following hardness results depend on the presence of an IR constraint. We first study the case where the objective is $\sum_{i=1}^{n} \pi_i$, that is, the designer is only interested in maximizing the total payment, and show that the problem of designing an optimal mechanism in this case is NP-complete.

Theorem 39 *The AMD problem for designing deterministic mechanisms is NP-complete, even*

when the objective is to maximize total payments made (under an IR constraint), there is only a single agent, and the probability distribution over Θ is uniform. (Membership in NP is guaranteed only if the number of agents is constant.)

Proof: The proof of membership in NP for a constant number of agents is similar to previous proofs. To show NP-hardness, we reduce an arbitrary MINSAT instance to the following automated mechanism design instance. Let the agent's type set be $\Theta = \{\theta_c : c \in C\} \cup \{\theta_v : v \in V\}$, where C is the set of clauses in the MINSAT instance, and V is the set of variables. Let the probability distribution over these types be uniform. Let the outcome set be $O = \{o_0\} \cup \{o_c : c \in C\} \cup \{o_l : l \in L\}$, where L is the set of literals, that is, $L = \{+v : v \in V\} \cup \{-v : v \in V\}$. Let the notation $v(l) = v$ denote that v is the variable corresponding to the literal l, that is, $l \in \{+v, -v\}$. Let $l \in c$ denote that the literal l occurs in clause c. Then, let the agent's utility function be given by $u(\theta_c, o_l) = |\Theta| + 1$ for all $l \in L$ with $l \in c$; $u(\theta_c, o_l) = 0$ for all $l \in L$ with $l \notin c$; $u(\theta_c, o_c) = |\Theta| + 1$; $u(\theta_c, o_{c'}) = 0$ for all $c' \in C$ with $c \neq c'$; $u(\theta_v, o_l) = |\Theta|$ for all $l \in L$ with $v(l) = v$; $u(\theta_v, o_l) = 0$ for all $l \in L$ with $v(l) \neq v$; $u(\theta_v, o_c) = 0$ for all $c \in C$. The goal of the automated mechanism design instance is $G = |\Theta| + \frac{|C|-K}{|\Theta|}$, where K is the goal of the MINSAT instance. We show the instances are equivalent. First, suppose there is a solution to the MINSAT instance. Let the assignment of truth values to the variables in this solution be given by the function $f : V \to L$ (where $v(f(v)) = v$ for all $v \in V$). Then, for every $v \in V$, let $o(\theta_v) = o_{f(v)}$ and $\pi(\theta_v) = |\Theta|$. For every $c \in C$, let $o(\theta_c) = o_c$; let $\pi(\theta_c) = |\Theta|+1$ if c is not satisfied in the MINSAT solution, and $\pi(\theta_c) = |\Theta|$ if c is satisfied. It is straightforward to check that the IR constraint is satisfied. We now check that the agent has no incentive to misreport. If the agent's type is some θ_v, then any other report will give it an outcome that is no better, for a payment that is no less, so it has no incentive to misreport. If the agent's type is some θ_c where c is a satisfied clause, again, any other report will give it an outcome that is no better, for a payment that is no less, so it has no incentive to misreport. The final case to check is where the agent's type is some θ_c where c is an unsatisfied clause. In this case, we observe that for none of the types, reporting it leads to an outcome o_l for a literal $l \in c$, precisely because the clause is not satisfied in the MINSAT instance. Because also, no type besides θ_c leads to the outcome o_c, reporting any other type will give an outcome with utility 0, while still forcing a payment of at least $|\Theta|$ from the agent. Clearly the agent is better off reporting truthfully, for a total utility of 0. This establishes that the agent never has an incentive to misreport. Finally, we show that the goal is reached. If s is the number of satisfied clauses in the MINSAT solution (so that $s \leq K$), the expected payment from this mechanism is $\frac{|V||\Theta|+s|\Theta|+(|C|-s)(|\Theta|+1)}{|\Theta|} \geq \frac{|V||\Theta|+K|\Theta|+(|C|-K)(|\Theta|+1)}{|\Theta|} = |\Theta| + \frac{|C|-K}{|\Theta|} = G$. So there is a solution to the automated mechanism design instance.

Now suppose there is a solution to the automated mechanism design instance, given by an outcome function o and a payment function π. First, suppose there is some $v \in V$ such that $o(\theta_v) \notin \{o_{+v}, o_{-v}\}$. Then the utility that the agent derives from the given outcome for this type is 0, and hence, by IR, no payment can be extracted from the agent for this type. Because, again by IR, the maximum payment that can be extracted for any other type is $|\Theta| + 1$, it follows that the maximum expected payment that could be obtained is at most $\frac{(|\Theta|-1)(|\Theta|+1)}{|\Theta|} < |\Theta| < G$, contradicting that this is a solution to the automated mechanism design instance. It follows that in the solution to the automated mechanism design instance, for every $v \in V$, $o(\theta_v) \in \{o_{+v}, o_{-v}\}$. We

can interpret this as an assignment of truth values to the variables: v is set to *true* if $o(\theta_v) = o_{+v}$, and to *false* if $o(\theta_v) = o_{-v}$. We claim this assignment is a solution to the MINSAT instance. By the IR constraint, the maximum payment we can extract from any type θ_v is $|\Theta|$. Because there can be no incentives for the agent to report falsely, for any clause c satisfied by the given assignment, the maximum payment we can extract for the corresponding type θ_c is $|\Theta|$. (For if we extracted more from this type, the agent's utility in this case would be less than 1; and if v is the variable satisfying c in the assignment, so that $o(\theta_v) = o_l$ where l occurs in c, then the agent would be better off reporting θ_v instead of the truthful report θ_c, to get an outcome worth $|\Theta| + 1$ to it while having to pay at most $|\Theta|$.) Finally, for any unsatisfied clause c, by the IR constraint, the maximum payment we can extract for the corresponding type θ_c is $|\Theta|+1$. It follows that the expected payment from our mechanism is at most $\frac{V|\Theta|+s|\Theta|+(|C|-s)(|\Theta|+1)}{\Theta}$, where s is the number of satisfied clauses. Because our mechanism achieves the goal, it follows that $\frac{V|\Theta|+s|\Theta|+(|C|-s)(|\Theta|+1)}{\Theta} \geq G$, which by simple algebraic manipulations is equivalent to $s \leq K$. So there is a solution to the MINSAT instance. ■

Finally, we study the case where the designer is self-interested and is not interested in payments made (that is, the objective is some function $g : O \to \mathbb{R}$). In this case, if the designer is allowed to use payments, then the designer can always choose her most preferred outcome by giving the agents an amount large enough to compensate them for the choice of this outcome, thereby not breaking the IR constraint. However, the case where the designer is not allowed to use payments is more complex, as the following theorem shows:

Theorem 40 *The AMD problem for designing deterministic mechanisms without payments is NP-complete, even when the designer is self-interested (but faces an IR constraint), there is only a single agent, and the probability distribution over Θ is uniform. (Membership in NP is guaranteed only if the number of agents is constant.)*

Proof: The proof of membership in NP for a constant number of agents is similar to previous proofs. To show NP-hardness, we reduce an arbitrary MINSAT instance to the following automated mechanism design instance. Let the agent's type set be $\Theta = \{\theta_c : c \in C\} \cup \{\theta_v : v \in V\}$, where C is the set of clauses in the MINSAT instance, and V is the set of variables. Let the probability distribution over these types be uniform. Let the outcome set be $O = \{o_0\} \cup \{o_c : c \in C\} \cup \{o_l : l \in L\} \cup \{o^*\}$, where L is the set of literals, that is, $L = \{+v : v \in V\} \cup \{-v : v \in V\}$. Let the notation $v(l) = v$ denote that v is the variable corresponding to the literal l, that is, $l \in \{+v, -v\}$. Let $l \in c$ denote that the literal l occurs in clause c. Then, let the agent's utility function be given by $u(\theta_c, o_l) = 2$ for all $l \in L$ with $l \in c$; $u(\theta_c, o_l) = -1$ for all $l \in L$ with $l \notin c$; $u(\theta_c, o_c) = 2$; $u(\theta_c, o_{c'}) = -1$ for all $c' \in C$ with $c \neq c'$; $u(\theta_c, o^*) = 1$; $u(\theta_v, o_l) = 1$ for all $l \in L$ with $v(l) = v$; $u(\theta_v, o_l) = -1$ for all $l \in L$ with $v(l) \neq v$; $u(\theta_v, o_c) = -1$ for all $c \in C$; $u(\theta_v, o^*) = -1$. Let the designer's objective function be given by $g(o^*) = |\Theta| + 1$; $g(o_l) = |\Theta|$ for all $l \in L$; $g(o_c) = |\Theta|$ for all $c \in C$. The goal of the automated mechanism design instance is $G = |\Theta| + \frac{|C|-K}{|\Theta|}$, where K is the goal of the MINSAT instance. We show the instances are equivalent. First, suppose there is a solution to the MINSAT instance. Let the assignment of truth values to the variables in this solution be given by the function $f : V \to L$ (where $v(f(v)) = v$ for all $v \in V$). Then, for every $v \in V$, let $o(\theta_v) = o_{f(v)}$. For every $c \in C$ that is satisfied in the MINSAT solution, let

$o(\theta_c) = o_c$; for every unsatisfied $c \in C$, let $o(\theta_c) = o^*$. It is straightforward to check that the IR constraint is satisfied. We now check that the agent has no incentive to misreport. If the agent's type is some θ_v, it is getting the maximum utility for that type, so it has no incentive to misreport. If the agent's type is some θ_c where c is a satisfied clause, again, it is getting the maximum utility for that type, so it has no incentive to misreport. The final case to check is where the agent's type is some θ_c where c is an unsatisfied clause. In this case, we observe that for none of the types, reporting it leads to an outcome o_l for a literal $l \in c$, precisely because the clause is not satisfied in the MINSAT instance. Because also, no type leads to the outcome o_c, there is no outcome that the mechanism ever selects that would give the agent utility greater than 1 for type θ_c, and hence the agent has no incentive to report falsely. This establishes that the agent never has an incentive to misreport. Finally, we show that the goal is reached. If s is the number of satisfied clauses in the MINSAT solution (so that $s \leq K$), then the expected value of the designer's objective function is $\frac{|V||\Theta|+s|\Theta|+(|C|-s)(|\Theta|+1)}{|\Theta|} \geq \frac{|V||\Theta|+K|\Theta|+(|C|-K)(|\Theta|+1)}{|\Theta|} = |\Theta| + \frac{|C|-K}{|\Theta|} = G$. So there is a solution to the automated mechanism design instance.

Now suppose there is a solution to the automated mechanism design instance, given by an outcome function o. First, suppose there is some $v \in V$ such that $o(\theta_v) \notin \{o_{+v}, o_{-v}\}$. The only other outcome that the mechanism is allowed to choose under the IR constraint is o_0. This has an objective value of 0, and because the highest value the objective function ever takes is $|\Theta| + 1$, it follows that the maximum expected value of the objective function that could be obtained is at most $\frac{(|\Theta|-1)(|\Theta|+1)}{|\Theta|} < |\Theta| < G$, contradicting that this is a solution to the automated mechanism design instance. It follows that in the solution to the automated mechanism design instance, for every $v \in V$, $o(\theta_v) \in \{o_{+v}, o_{-v}\}$. We can interpret this as an assignment of truth values to the variables: v is set to *true* if $o(\theta_v) = o_{+v}$, and to *false* if $o(\theta_v) = o_{-v}$. We claim this assignment is a solution to the MINSAT instance. By the above, for any type θ_v, the value of the objective function in this mechanism will be $|\Theta|$. For any clause c satisfied by the given assignment, the value of the objective function in the case where the agent reports type θ_c will be at most $|\Theta|$. (This is because we cannot choose the outcome o^* for such a type, as in this case the agent would have an incentive to report θ_v instead, where v is the variable satisfying c in the assignment (so that $o(\theta_v) = o_l$ where l occurs in c).) Finally, for any unsatisfied clause c, the maximum value the objective function can take in the case where the agent reports type θ_c is $|\Theta| + 1$, simply because this is the largest value the function ever takes. It follows that the expected value of the objective function for our mechanism is at most $\frac{V|\Theta|+s|\Theta|+(|C|-s)(|\Theta|+1)}{\Theta}$, where s is the number of satisfied clauses. Because our mechanism achieves the goal, it follows that $\frac{V|\Theta|+s|\Theta|+(|C|-s)(|\Theta|+1)}{\Theta} \geq G$, which by simple algebraic manipulations is equivalent to $s \leq K$. So there is a solution to the MINSAT instance. ■

6.4 Linear and mixed integer programming approaches

In this section, we describe how the problem of designing an optimal *randomized* mechanism can be cast as a linear programming problem. As we will show, the size of the linear program is exponential only in the number of agents, and because linear programs can be solved in polynomial

time [Khachiyan, 1979], this implies that the problem of designing an optimal[6] randomized mechanism is in P if the number of agents is a constant.

Theorem 41 *With a constant number of agents, the optimal randomized mechanism can be found in polynomial time using linear programming, both with and without payments, both for* ex post *and* ex interim *IR, and both for implementation in dominant strategies and for implementation in Bayes-Nash equilibrium—even if the types are correlated (that is, an agent's type tells him something about the other agents' types).*

Proof: Because linear programs can be solved in polynomial time, all we need to show is that the number of variables and equations in our program is polynomial for any constant number of agents—that is, exponential only in n. Throughout, for purposes of determining the size of the linear program, let $T = \max_i\{|\Theta_i|\}$. The variables of our linear program will be the probabilities $(p(\theta_1, \theta_2, \ldots, \theta_n))(o)$ (at most $T^n|O|$ variables) and the payments $\pi_i(\theta_1, \theta_2, \ldots, \theta_n)$ (at most nT^n variables). (We show the linear program for the case where payments are possible; the case without payments is easily obtained from this by simply omitting all the payment variables in the program, or by adding additional constraints forcing the payments to be 0.)

First, we show the IR constraints. For *ex post* IR, we add the following (at most nT^n) constraints to the LP:

• For every $i \in \{1, 2, \ldots, n\}$, and for every $(\theta_1, \theta_2, \ldots, \theta_n) \in \Theta_1 \times \Theta_2 \times \ldots \times \Theta_n$, we add

$$(\sum_{o \in O} (p(\theta_1, \theta_2, \ldots, \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, \ldots, \theta_n) \geq 0.$$

For *ex interim* IR, we add the following (at most nT) constraints to the LP:

• For every $i \in \{1, 2, \ldots, n\}$, for every $\theta_i \in \Theta_i$, we add

$$\sum_{\theta_1, \ldots, \theta_n} \gamma(\theta_1, \ldots, \theta_n | \theta_i)((\sum_{o \in O} (p(\theta_1, \theta_2, \ldots, \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, \ldots, \theta_n)) \geq 0.$$

Now, we show the solution concept constraints. For implementation in dominant strategies, we add the following (at most nT^{n+1}) constraints to the LP:

• For every $i \in \{1, 2, \ldots, n\}$, for every
$(\theta_1, \theta_2, \ldots, \theta_i, \ldots, \theta_n) \in \Theta_1 \times \Theta_2 \times \ldots \times \Theta_n$, and for every alternative type report $\hat{\theta}_i \in \Theta_i$, we add the constraint

$$(\sum_{o \in O} (p(\theta_1, \theta_2, \ldots, \theta_i, \ldots, \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, \ldots, \theta_i, \ldots, \theta_n) \geq$$
$$(\sum_{o \in O} (p(\theta_1, \theta_2, \ldots, \hat{\theta}_i, \ldots, \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, \ldots, \hat{\theta}_i, \ldots, \theta_n).$$

Finally, for implementation in Bayes-Nash equilibrium, we add the following (at most nT^2) constraints to the LP:

[6]Since linear programs allow for an objective, we can search for the optimal mechanism rather than only solve the decision variant (does a mechanism with objective value at least G exist?) of the problem.

• For every $i \in \{1, 2, ..., n\}$, for every $\theta_i \in \Theta_i$, and for every alternative type report $\hat{\theta}_i \in \Theta_i$, we add the constraint

$$\sum_{\theta_1,...,\theta_n} \gamma(\theta_1, ..., \theta_n|\theta_i)((\sum_{o \in O} (p(\theta_1, \theta_2, ..., \theta_i, ..., \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, ..., \theta_i, ..., \theta_n)) \geq$$
$$\sum_{\theta_1,...,\theta_n} \gamma(\theta_1, ..., \theta_n|\theta_i)((\sum_{o \in O} (p(\theta_1, \theta_2, ..., \hat{\theta}_i, ..., \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, ..., \hat{\theta}_i, ..., \theta_n)).$$

All that is left to do is to give the expression the designer is seeking to maximize, which is:

• $\sum_{\theta_1,...,\theta_n} \gamma(\theta_1, ..., \theta_n)((\sum_{o \in O} (p(\theta_1, \theta_2, ..., \theta_i, ..., \theta_n))(o)g(o)) + \sum_{i=1}^{n} \pi_i(\theta_1, \theta_2, ..., \theta_n))$.

As we indicated, the number of variables and constraints is exponential only in n, and hence the linear program is of polynomial size for constant numbers of agents. Thus the problem is solvable in polynomial time. ■

By forcing all the probability variables to be either 0 or 1, thereby changing the linear program into a mixed integer program, we can solve for the optimal deterministic mechanism. (We note that solving a mixed integer program is NP-complete.) Our general-purpose automated mechanism design solver consists of running CPLEX (a commercial solver) on these linear or mixed integer programs.

6.5 Initial applications

The only example application that we have seen so far is the divorce settlement setting from Section 6.2. In this section, we apply AMD to some domains that are more commonly studied in mechanism design: optimal auctions and mechanisms for public goods.

6.5.1 Optimal auctions

In this subsection we show how AMD can be used to design auctions that maximize the seller's expected revenue (so-called *optimal* auctions). In many auction settings, the seller would like to design the rules of the auction to accomplish this. However, as we briefly mentioned in Chapter 4, in general settings this is a known difficult mechanism design problem; for one, it is much more difficult than designing a mechanism that allocates the goods efficiently (among bidders with quasi-linear preferences, *ex post* efficiency and IR can be accomplished in dominant strategies using the VCG mechanism).

We first study auctioning off a single good, and show that AMD reinvents a known landmark optimal auction mechanism, the Myerson auction, for the specific instance that we study. (Of course, it does not derive the general form of the Myerson auction, which can be applied to any single-item instance: AMD necessarily only solves the instance at hand.) We then move to multi-item (combinatorial) auctions, where the optimal auction has been unknown in the literature to date. We show that AMD can design optimal auctions for this setting as well.

An optimal 2-bidder, 1-item auction

We first show how automated mechanism design can rederive known results in optimal single-item auction design. Say there is one item for sale. The auctioneer can award it to any bidder, or not award it (and say the auctioneer's valuation for the good is 0). There are two bidders, 1 and 2. For each of them, their distribution of valuations is uniform over $\{0, 0.25, 0.5, 0.75, 1\}$.

In designing the auction automatically, we required ex-interim IR and implementation in Bayes-Nash equilibrium. Randomization was allowed (although in this setting, it turned out that the probabilities were all 0 or 1). The allocation rule of the mechanism generated by the solver is as follows. If both bid below 0.5, do not award the item; otherwise, give the item to the highest bidder (a specific one of them in the case of a tie). This is effectively[7] the celebrated *Myerson auction* [Myerson, 1981] (although the Myerson auction was originally derived for a continuous valuation space). So, AMD quickly reinvented a landmark mechanism from 1981. (Although it should be noted that it invented it for a special case, and did not derive the general characterization. Also, it did not invent the *question* of optimal auction design.)

Multi-item (combinatorial) auctions

We now move to combinatorial auctions where there are multiple goods for sale. The design of a mechanism for this setting that maximizes the seller's expected revenue is a recognized open research problem [Avery and Hendershott, 2000; Armstrong, 2000; Vohra, 2001]. The problem is open even if there are only two goods for sale. (The two-good case with a very special form of complementarity and no substitutability has been solved recently [Armstrong, 2000].) We show that AMD can be used to generate optimal combinatorial auctions.

In our first combinatorial auction example, two items, A and B, are for sale. The auctioneer can award each item to any bidder, or not award it (and the auctioneer's valuation is 0). There are two bidders, 1 and 2, each of whom has four possible, equally likely types: LL, HL, LH, and HH. The type indicates whether each item is strongly desired or not; for instance, the type HL indicates that the bidder strongly desires the first item, but not the second. Getting an item that is strongly desired gives utility 2; getting one that is not strongly desired gives utility 1. The utilities derived from the items are simply additive (no substitution or complementarity effects), with the exception of the case where the bidder has the type HH. In this case there is a complementarity bonus of 2 for getting both items (thus, the total utility of getting both items is 6). (One way to interpret this is as follows: a bidder will sell off any item it wins and does not strongly desire, on a market where it is a price taker, so that there are no substitution or complementarity effects with such an item.)

In designing the auction, we required ex-interim IR and implementation in Bayes-Nash equilibrium. Randomization was allowed (although in this setting, it turned out that the probabilities were all 0 or 1). The objective to maximize was the expected payments from the bidders to the seller. The mechanism generated by the solver has the following allocation rule: 1. If one bidder bid LL, then the other bidder gets all the items he bid high on, and all the other items (that both bid low on) are not awarded. 2. If exactly one bidder bid HH, that bidder gets both items. If both bid HH, bidder 1 gets both items. 3. If both bidders bid high on only one item, and they did not bid high on

[7] The payment rule generated is slightly different, because CPLEX chooses to distribute the payments slightly differently across different type vectors.

the same item, each bidder gets his preferred item. 4. If both bidders bid high on only one item, and they bid high on the same item, bidder 2 gets the preferred item, and bidder 1 gets the other item.

	LL	LH	HL	HH
LL	0, 0	0, 2	2, 0	2, 2
LH	0, 1	1, 2	2, 1	2, 2
HL	1, 0	1, 2	2, 1	2, 2
HH	1, 1	1, 1	1, 1	1, 1

The allocation rule in the optimal combinatorial auction. The row indicates bidder 1's type, the column bidder 2's type. i, j indicates that item A goes to bidder i, and item B to bidder j. (0 means the item is not awarded to anyone.)

It is interesting to observe that suboptimal allocations occur only when one bidder bids LL and the other other does not bid HH. All the inefficiency stems from not awarding items, never from allocating items to a suboptimal bidder.

The expected revenue from the mechanism is 3.9375. For comparison, the expected revenue from the VCG mechanism is only 2.6875. It is interesting to view this in light of a recent result that the VCG mechanism is asymptotically (as the number of bidders goes to infinity) optimal in multi-item auctions, that is, it maximizes revenue in the limit [Monderer and Tennenholtz, 1999].[8] Apparently the auction will need to get much bigger (have more bidders) before no significant fraction of the revenue is lost by using the VCG mechanism. (Of course, this is only a single instance—future research may determine how much revenue is typically lost by the VCG mechanism for instances of this size, as well as determine how this changes when the instances become somewhat larger.)

We now move on to designing a bigger auction, again with 2 items, but now with 3 bidders (for a total of 16 possible allocations of items) and a bigger type space. Again, the bidders can have a high or low type for each item, resulting in a utility for that item alone of 3 or 1, respectively; part of their type now also includes whether the items have complementarity or substitutability to them, resulting in a total of 8 types per bidder—that is, $8^3 = 512$ type vectors (possible joint preference revelations by the bidders). In the case where the items have substitutability, the utility of getting both items is the sum of the items' individual values, minus 0.2 times the value of the lesser valued item.[9] In the case of complementarity, 0.2 times the value of the lesser-valued item is *added*.

This is the only instance in this section where CPLEX took more than 0.00 seconds to solve the instance. (It took 5.90 seconds.) The optimal auction generated has an expected revenue of 5.434. The allocation rule generated (an 8x8x8 table) is too large to present, but we point out some interesting properties of the optimal auction generated nonetheless:

1. Sometimes, items are again not awarded, for example, when two bidders report a low valuation for both items and the remaining bidder does not report a high valuation on both items;

[8] This result is particularly easy to prove in a discretized setting such as the one we are considering. The following sketches the proof. As the number of bidders grows, it becomes increasingly likely that for each winning bid, there is another submitted bid that is exactly identical, but not accepted. If this is the case, the VCG payment for the winning bid is exactly the value of that bid, and thus the VCG mechanism extracts the maximum possible payment. (This is also roughly the line of reasoning taken in the more general result [Monderer and Tennenholtz, 1999].)

[9] Subtracting a fraction from the lesser valued item guarantees free disposal, *i.e.* additional items cannot make a bidder worse off.

2. Randomization now *does* occur, for instance sometimes (but not always) when one item is valued lowly by everyone and two of the three value the other item highly (the randomization is over which of the two gets the desired item);

3. The optimal auction takes the complementarity and substitutability into account, for instance by doing the following. When one bidder bids high on both items and the other two each bid high on one item (not the same one), then the mechanism awards the items to the first bidder if that bidder revealed complementarity, but to the other bidders if the first bidder revealed substitutability. (Each one gets his/her desired item.)

It turns out, however, that the optimal deterministic mechanism generated for this instance has the same expected revenue (5.434). Thus, one may wonder if randomization is ever necessary to create optimal combinatorial auctions. The following example shows that there are indeed instances of the optimal combinatorial auction design problem where randomized mechanisms can perform strictly better than any deterministic mechanism.

In this example, there are two items A and B for sale, and there is only a single bidder (so that it does not matter which solution concept and which IR notion we use). There are three types: type α (occurring with probability 0.3) indicates that the bidder has a utility of 1 for receiving either $\{A\}$ or $\{A, B\}$, and 0 otherwise; type β (occurring with probability 0.3) indicates that the bidder has a utility of 1 for receiving either $\{B\}$ or $\{A, B\}$, and 0 otherwise; and type $\alpha\beta$ (occurring with probability 0.4) indicates that the bidder has a utility of 0.75 for receiving either $\{A\}$, $\{B\}$, or $\{A, B\}$, and 0 otherwise.

The optimal randomized mechanism generated by the solver allocates $\{A\}$ to the bidder if the bidder reports α; $\{B\}$ if the bidder reports β; and $\{A\}$ with probability 0.75, and $\{B\}$ with probability 0.25, if the bidder reports $\alpha\beta$. The payment in each case is the agent's entire valuation (1 for types α and β, and 0.75 for type $\alpha\beta$). The resulting expected revenue is 0.9.

By contrast, the optimal deterministic mechanism generated by the solver allocates $\{A\}$ to the bidder if the bidder reports α; $\{A, B\}$ if the bidder reports β; and $\{A\}$ if the bidder reports $\alpha\beta$. The payment is 0.75 for type α, 1 for type β, and 0.75 for type $\alpha\beta$. The resulting expected revenue is 0.825.

This example demonstrates how AMD can be used to disprove conjectures in mechanism design: the conjecture that one can restrict attention to deterministic mechanisms in the design of optimal combinatorial auctions is disproved by an example where the optimal randomized mechanism produced by the solver is strictly better than the optimal deterministic one.

6.5.2 Public goods problems

As another example application domain, we now turn to public good problems. A *public good* is a good from which many agents can benefit simultaneously; the good is said to be *nonexcludable* if we cannot prevent any agents from obtaining this benefit, given that we produce the good. Examples of nonexcludable public goods include clean air, national defense, pure research, *etc.*

A typical mechanism design problem that arises is that a certain amount of money is required to construct or acquire the (nonexcludable) public good, and this money must be collected from the agents that may benefit from it. However, how much the good is worth to each agent is information

that is private to that agent, and we cannot obtain a larger payment from an agent than what the agent claims the good is worth to him (an individual rationality constraint). This leads to the potential problem of *free riders* who report very low values for the good in the hope that other agents will value the good enough for it to still be produced. Formally, 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 $\sum_i \pi_i \geq c$, where c is the cost of producing the good. Results similar to the Myerson-Satterthwaite impossibility theorem can be proved here to show that even in quite simple settings, there is no mechanism that is *ex post* efficient, *ex post* budget balanced, *ex-interim* individually rational, and BNE incentive-compatible. In fact, Theorem 36 from Chapter 5 shows exactly this.

The advantage of applying AMD in this setting is that we do not desire to design a mechanism for general (quasilinear) preferences, but merely for the specific mechanism design problem instance at hand. In some settings this may allow one to circumvent the impossibility entirely, and in all settings it minimizes the pain entailed by the impossibility.

Building a bridge

Two agents are deciding whether to build a good that will benefit both (say, a bridge). The bridge, if it is to be built, must be financed by the payments made by the agents. Building the bridge will cost 6. The agents have the following type distribution: with probability .4, agent 1 will have a low type and value the bridge at 1. With probability .6, agent 1 will have a high type and value the bridge at 10. Agent 2 has a low type with probability .6 and value the bridge at 2; with probability .4, agent 2 will have a high type and value the bridge at 11. (Thus, agent 2 cares for the bridge more in both cases, but agent 1 is more likely to have a high type.)

We used AMD to design a randomized, dominant-strategies incentive compatible, *ex post* IR mechanism that is as efficient as possible—*taking into account unnecessary payments ("money burning") as a loss in efficiency*. The optimal mechanism generated by our AMD implementation has the following outcome function (here the entries of the matrix indicate the probability of building the bridge in each case):

	Low	High
Low	0	.67
High	1	1

The payment function is as follows (here a, b gives the payments of agents 1 and 2, respectively):

	Low	High
Low	0, 0	.67, 3.33
High	4, 2	4, 2

The payments in the case where agent 1 bids low but agent 2 bids high are the *expected payments* (as we argued before, risk-neutral agents only care about this); the agents will need to pay more than this when the good is actually built, but can pay less when it is not. (The constraints on the expected payments in the linear program are set so that the good can always be afforded when it is built.)

It is easy to see that no money is burned: all the money the agents pay goes towards building the bridge. However, we do not always build the bridge when this is socially optimal—namely, when the second agent has a high type (which is enough to justify building the bridge) we do not always build the bridge.

If we relax our solution concept to implementation in Bayes-Nash equilibrium, however, we get a mechanism with the following outcome function:

	Low	High
Low	0	1
High	1	1

The payment function is now as follows:

	Low	High
Low	0, 0	0, 6
High	4, 2	.67, 5.33

Again, no money is burned, but now also, the optimal outcome is always chosen. Thus, with Bayes-Nash equilibrium incentive compatibility, our mechanism achieves everything we hope for in this instance—even though the impossibility result shows that this is not possible for *all* instances.

Building a bridge and/or a boat

Now let us move to the more complex public goods setting where two goods could be built: a bridge and a boat. There are 4 different outcomes corresponding to which goods are built: None, Boat, Bridge, Boat and Bridge. The boat costs 1 to build, the bridge 2, and building both thus costs 3.

The two agents each have one of four different types: None, Boat Only, Bridge Only, Boat or Bridge. These types indicate which of the two possible goods would be helpful to the agent (for instance, maybe one agent would only be helped by a bridge because this agent wants to take the car to work, which will not fit on the boat). All types are equally likely; if something is built which is useful to a agent (given that agent's type), the agent gets a utility of 2, otherwise 0.

We used AMD to design the optimal randomized dominant-strategy mechanism that is ex post IR, and as ex post efficient as possible—taking into account money burning as a loss in efficiency. The mechanism has the following outcome function, where a vector (a, b, c, d) indicates the probabilities for None, Boat, Bridge, Boat and Bridge, respectively.

	None	Boat	Bridge	Either
None	(1,0,0,0)	(0,1,0,0)	(1,0,0,0)	(0,1,0,0)
Boat	(.5,.5,0.0)	(0,1,0,0)	(0,.5,0,.5)	(0,1,0,0)
Bridge	(1,0,0,0)	(0,1,0,0)	(0,0,1,0)	(0,0,1,0)
Either	(.5,.5,0.0)	(0,1,0,0)	(0,0,1,0)	(0,1,0,0)

The (expected) payment function is as follows:

	None	Boat	Bridge	Either
None	0,0	0,1	0,0	0,1
Boat	.5,0	0,1	1,1	0,1
Bridge	0,0	0,1	1,1	1,1
Either	.5,0	0,1	1,1	0,1

Again, no money is burned, but we do not always build the public goods that are socially optimal—for example, sometimes nothing is built although the boat would have been useful to someone.

6.6 Scalability experiments

To assess the scalability of the automated mechanism design approach in general, we generated random instances of the automated mechanism design problem. Each agent, for each of its types, received a utility for each outcome that was uniformly randomly chosen from the integers $0, 1, 2, \ldots, 99$. (All random draws were independent.) Real-world automated mechanism design instances are likely to be more structured than this (for example, in allocation problems, if one agent is happy with an outcome, this is because it was allocated a certain item that it wanted, and thus other agents who wanted the item will be less happy); such special structure can typically be taken advantage of in computing the optimal mechanism, even by nonspecialized algorithms. For instance, a random instance with 3 agents, 16 outcomes, 8 types per agent, with payment maximization as its goal, ex-interim IR, implementation in Bayes-Nash equilibrium, where randomization is allowed, takes 14.28 seconds to solve on average in our implementation. The time required to compute the last optimal combinatorial auction from Section 6.5, which had exactly the same parameters (but much more structure in the utility functions), compares (somewhat) favorably to this at 5.90 seconds.

We are now ready to present the scalability results. For every one of our experiments, we consider both implementation in dominant strategies and implementation in Bayes-Nash equilibrium. We also consider both the problem of designing a deterministic mechanism and that of designing a randomized mechanism. All the other variables that are not under discussion in a particular experiment are fixed at a default value (4 agents, 4 outcomes, 4 types per agent, no IR constraint, no payments, social welfare is the objective); these default values are chosen to make the problem hard enough for its runtime to be interesting. Experiments taking longer than 6 hours were cancelled, as well as experiments where the LP size was greater than 400MB. CPLEX does not provide runtime information more detailed than centiseconds, which is why we do not give the results with a constant number of significant digits, but rather all the digits we have.

The next table shows that the runtime increases fairly sharply with the number of agents. Also (as will be confirmed by all the later experiments), implementation in dominant strategies is harder than implementation in BNE, and designing deterministic mechanisms is harder than designing randomized mechanisms. (The latter part is consistent with the transition from NP-completeness to solvability in polynomial time by allowing for randomness in the mechanism (Sections 6.3 and 6.4).)

#agents	D/DSE	R/DSE	D/BNE	R/BNE
2	.02	.00	.00	.00
3	.04	.00	.05	.01
4	8.32	1.32	1.68	.06
5	709.85	48.19	10.47	.52

The time (in seconds) required to solve randomly generated AMD instances for different numbers of agents, for deterministic (D) or randomized (R) mechanisms, with implementation in dominant strategies (DSE) or Bayes-Nash equilibrium (BNE). All experiments had 4 outcomes and 4 types per agent, required no IR constraint, did not allow for payments, and had social welfare as the objective.

The next table shows that the runtime tends to increase with the number of outcomes, but not at all sharply.

#outcomes	D/DSE	R/DSE	D/BNE	R/BNE
2	.07	.07	.04	.03
3	.36	.08	.46	.05
4	8.32	1.32	1.68	.06
5	10.91	.59	.69	.07

The next table shows that the runtime increases fairly sharply with the number of types per agent.

#types	D/DSE	R/DSE	D/BNE	R/BNE
2	.00	.00	.00	.00
3	.04	.01	.30	.01
4	8.32	1.32	1.68	.06
5	563.73	14.33	36.60	.21

Because the R/BNE case scales reasonably well in each setting, we increased the numbers of agents, outcomes, and types further for this case to test the limits of our implementation. Our initial implementation requires the linear program to be written out explicitly, and thus space eventually became the bottleneck for scaling in agents and types. ("*" indicates that the LP size exceeded 400MB.) Mature techniques exist for linear programming when the LP is too large to write down, and future implementations could make use of these techniques.

#	agents	outcomes	types
6	4.39	.07	.88
7	33.32	.07	1.91
8	*	.09	4.52
10	*	.11	22.05
12	*	.13	67.74
14	*	.13	*
100	*	1.56	*

The next table shows that the impact of IR constraints on runtime is entirely negligible.

IR constraint	D/DSE	R/DSE	D/BNE	R/BNE
None	8.32	1.32	1.68	.06
Ex post	8.20	1.38	1.67	.12
Ex interim	8.11	1.42	1.65	.11

The next table studies the effects of allowing for payments and changing the objective. Allowing for payments (without taking the payments into account) in social welfare maximization reduces the runtime. This appears consistent with the fact that for this setting, a general (and easy-to-compute) mechanism exists that always obtains the maximum social welfare—the VCG mechanism. However, this speedup disappears when we start taking the payments into account. Interestingly, payment maximization appears to be much harder than social welfare maximization. In particular, in one case (designing a deterministic mechanism without randomization), an optimal mechanism had not been constructed after 6 hours!

Objective	D/DSE	R/DSE	D/BNE	R/BNE
SW (1)	8.20	1.38	1.67	.12
SW (2)	.41	.14	.92	.10
SW (3)	7.98	.51	4.44	.10
π	-	1.89	84.66	3.47

SW=social welfare (1) without payments, (2) with payments that are not taken into account in social welfare calculations, (3) with payments that are taken into account in social welfare calculations; π=payment maximization.

The sizes of the instances that we can solve may not appear very impressive when compared with the sizes of (for instance) combinatorial auctions currently being studied in the literature. While this is certainly true, we emphasize that 1. We are studying a much more difficult problem than the auction clearing problem: we are *designing* the mechanism, rather than executing it; 2. AMD is still in its infancy, and it is likely that future (possibly approximate) approaches will scale to much larger instances; and 3. Although many real-world instances are very large, there are also many small ones. Moreover, the "small" instances may concern equally large dollar values as the large ones. For example, selling two masterpieces by Picasso in a combinatorial auction could create revenue comparable to that of selling a million plane tickets in a combinatorial auction.

6.7 An algorithm for single-agent settings

The definitions from Section 6.1 simplify significantly when applied to the setting where a deterministic mechanism without payments must be designed, with a single type-reporting agent. For one, the different possible IC (truthfulness) constraints differ only in what a type-reporting agent is assumed to know about other type-reporting agents' preferences and reports. Because in this setting, there are no other type-reporting agents, the different IC constraints coincide. The same is true for the IR (participation) constraints. We also do not need distributions over outcomes, or payment functions. The result is the following formal definition for our special case.

Definition 27 (AUTOMATED-MECHANISM-DESIGN (AMD)) *We are given a set of outcomes O, and a set of types Θ for the agent together with a probability distribution p over these types. Additionally we are given a utility function for the agent, $u : \Theta \times O \to \mathbb{R}$, and an objective function for the designer, $g : \Theta \times O \to \mathbb{R}$. We are asked to find an outcome function $o : \Theta \to O$ (a deterministic mechanism without payments) such that:*

1. *For every $\theta, \hat{\theta} \in \Theta$, $u(\theta, o(\theta)) \geq u(\theta, o(\hat{\theta}))$ (the IC constraint).*
2. *If there is an IR constraint, for every $\theta \in \Theta$, $u(\theta, o(\theta)) \geq 0$. (In this case there typically also is a* default outcome *o_0 with $u(\theta, o_0) = 0$ for all $\theta \in \Theta$.[10])*
3. *Subject to the previous constraints, the mechanism maximizes $\sum_{\theta \in \Theta} p(\theta) g(\theta, o(\theta))$.*

We note that by Theorem 37, even this specialized problem is NP-complete (even without the IR constraint, and even when the objective function is a social welfare function including another agent that does not report a type).

6.7.1 Application: One-on-one bartering

As an interlude, we first present an application. Consider the situation where two agents each have an initial endowment of goods. Each agent has a valuation for every subset of the m goods that the agents have together. It is possible that both agents can become better off as a result of trade. Suppose, however, that the agents cannot make any form of payment; all they can do is swap goods. This is known as *bartering*. Additionally, suppose that one agent (agent 1) is in the position of dictating the rules of the bartering process. Agent 1 can credibly say to agent 2, "we will barter by my rules, or not at all." This places agent 1 in the position of the mechanism designer, and corresponds to the following AMD problem. The set of outcomes is the set of all allocations of the goods (there are 2^m of them). Agent 2 is to report his preferences over the goods (the valuation that agent has for each subset), and on the basis of this report an outcome is chosen. This outcome function, which is selected by agent 1 beforehand, must be incentive compatible so that agent 2 has no incentive to misreport. Also, it must be individually rational, or agent 2 simply will not trade.[11] Under these constraints, agent 1 wishes to make the expected value of her own allocation under the mechanism as large as possible. The revelation principle justifies that restricting agent 1 to this approach comes at no loss to that agent.

Automatically generated mechanisms for this setting are likely to be useful in barter-based electronic marketplaces, such as mybarterclub.com, Recipco, and National Trade Banc.

We now return to computational aspects, but we will readdress the bartering problem in our experiments. We will postpone dealing with IR constraints for a few subections, and then return to this.

[10] We can set the utility of the default outcome to 0 without loss of generality, by normalizing the utility function. (From a decision-theoretic point of view it does not matter how utilities compare across types, because the agent always knows her own type and will not take utilities for other types into account in making any decision.)

[11] If agent 1 actually wants to make the rules so that there is no trade for a certain type report, she can simply make the original allocation the outcome for this type report; so there is no loss to agent 1 in designing the outcome function in such a way that agent 2 always wishes to participate in the mechanism.

6.7.2 Search over subsets of outcomes

In this subsection, we associate with each subset of outcomes a truthful mechanism for that set of outcomes; we then show that for some subset of outcomes, the truthful mechanism associated with that subset of outcomes is an optimal mechanism for the setting. Because the mechanism associated with a subset of outcomes is easy to compute, we can search over subsets of outcomes (of which there are $2^{|O|}$) rather than over all possible outcome functions (of which there are $|O|^{|\Theta|}$).[12]

We first define the outcome function (mechanism) o_X associated with a particular subset of the outcomes.

Definition 28 *For a given subset $X \subseteq O$, let $o_X(\theta)$ be (the lowest-indexed element of) $\arg\max_{\{o \in X : (\forall o' \in X) u(\theta,o) \geq u(\theta,o')\}} g(\theta, o)$. Let $v(X)$ be given by $\sum_{\theta \in \Theta} p(\theta) g(\theta, o_X(\theta))$.*

Intuitively, $o_X(\theta)$ is the outcome we wish to pick for type θ, if we (somehow) know that the set of other outcomes used in the mechanism is exactly X, and we wish to pick an outcome from X as well. $v(X)$ is the expected value of the objective function for the mechanism o_X, presuming that the agent reports truthfully. The next lemma shows that indeed, the agent has no incentive to report falsely.

Lemma 19 *For all $X \subseteq O$, o_X is truthful. (Thus, $v(X)$ is indeed the expected value of the objective function for it.)*

Proof: For any pair of types θ_1, θ_2, we have that $o_X(\theta_2) \in X$ because all outcomes ever chosen by o_X are in X; and thus that $u(\theta_1, o_X(\theta_1)) \geq u(\theta_1, o_X(\theta_2))$, because for any θ, $o_X(\theta)$ maximizes $u(\theta, \cdot)$ among outcomes $o \in X$. ■

The next lemma shows that for any subset X, the mechanism o_X dominates all mechanisms that use exactly the outcomes in X.

Lemma 20 *For any $X \subseteq O$, suppose that $o : \Theta \to X$ is a truthful mechanism making use only of outcomes in X, but using each outcome in X at least once—that is, $o(\Theta) = X$. Let its expected value of the objective function be $v_o = \sum_{\theta \in \Theta} p(\theta) g(\theta, o(\theta))$. Then $v(X) \geq v_o$.*

Proof: For any $\theta \in \Theta$, we must have that for any $o \in X$, $u(\theta, o(\theta)) \geq u(\theta, o)$—because there exists some $\theta' \in \Theta$ such that $o(\theta') = o$, and thus the agent can guarantee herself at least utility $u(\theta, o)$ by reporting θ'. But $o_X(\theta)$ maximizes $g(\theta, \cdot)$ among such outcomes. Thus, $g(\theta, o_X(\theta)) \geq g(\theta, o(\theta))$. It follows that $v(X) = \sum_{\theta \in \Theta} p(\theta) g(\theta, o_X(\theta)) \geq \sum_{\theta \in \Theta} p(\theta) g(\theta, o(\theta)) = v_o$. ■

It is not necessarily the case that $v(X) = v_o$ for some truthful o making use of all outcomes in X; for instance, there could be some outcome in X that has both a very low utility value and a very low

[12] In the case where $|O|$ is bigger than $|\Theta|$, we can restrict ourselves to outcome subsets of size at most $|\Theta|$, making our approach still more efficient than the straightforward brute search approach. For simplicity of presentation, in this section we will focus on settings where $|\Theta| > |O|$ (as is commonly the case).

objective value. Then o_X will not use this outcome, and thereby have a higher expected value of the objective function than any mechanism that does use it.

We are now ready to present the main theorem of this subsection, which states that the best o_X is indeed an optimal mechanism.

Theorem 42 $\max_{X \subseteq O} v(X)$ *is the maximum expected value of the objective over* all *mechanisms (that are deterministic and use no payments).* o_X *is an optimal mechanism (among mechanisms that are deterministic and use no payments) if* $X \in \arg\max_{X \subseteq O} v(X)$.

Proof: Consider an optimal truthful mechanism o,[13] and let X be the set of all outcomes it uses ($X = o(\Theta)$). By Lemma 19, o_X is truthful and $v(X)$ is the expected value of the objective function for it. By Lemma 20, we have $v(X) \geq v_o$ where v_o is the expected value of the objective function for o. ■

6.7.3 A heuristic and its admissibility

We now proceed to define an outcome function that is associated with two disjoint subsets X and Y of the outcomes; we will use this outcome function to compute an admissible heuristic for our search problem. The interpretation is as follows. In the process of constructing a mechanism of the kind described in the previous subsection, we successively label each outcome as "in" or "out", depending on whether we wish to include this outcome in the set that the eventual mechanism is associated with. X consists of the outcomes that we have already decided are "in"; Y consists of the outcomes that we have already decided are "out". To get an optimistic view of the mechanisms we may eventually arrive at from here, we assign to each type the outcome in $O - Y$ that gives us the highest objective value for that type (the mechanisms certainly will not use any outcome in Y), under the constraint that this outcome will make that type at least as well off as any outcome in X (because we have already decided that these are certainly "in", so we know this constraint must apply).

Definition 29 *For given subsets* $X, Y \subseteq O$, *let* $o_{X,Y}(\theta)$ *be (the lowest-indexed element of)* $\arg\max_{o \in O-Y : (\forall o' \in X) u(\theta,o) \geq u(\theta,o')} g(\theta, o)$. *Let* $v(X, Y)$ *be given by* $\sum_{\theta \in \Theta} p(\theta) g(\theta, o_{X,Y}(\theta))$

Outcome functions of this type do *not* necessarily constitute truthful mechanisms. (For instance, if X and Y are both the empty set, then $o_{X,Y}$ will simply choose the objective-maximizing outcome for each type.) Nevertheless, because we are merely trying to obtain an optimistic estimate, we compute $v(X, Y)$ as before, presuming the agents will report truthfully. The following theorem shows that $v(X, Y)$ is indeed admissible.

Theorem 43 *For any subsets* $X, Y \subseteq O$, *for any* $Z \subseteq O - X - Y$, *for any* $\theta \in \Theta$, *we have* $g(\theta, o_{X,Y}(\theta)) \geq g(\theta, o_{X \cup Z}(\theta))$; *and* $v(X, Y) \geq v(X \cup Z)$.

[13] Which, by the revelation principle, is an optimal mechanism.

Proof: Using the facts that $X \subseteq X \cup Z$ and $X \cup Z \subseteq O - Y$, we can conclude that $\{o \in X \cup Z : (\forall o' \in X \cup Z) u(\theta, o) \geq u(\theta, o')\} \subseteq \{o \in X \cup Z : (\forall o' \in X) u(\theta, o) \geq u(\theta, o')\} \subseteq \{o \in O - Y : (\forall o' \in X) u(\theta, o) \geq u(\theta, o)\}$. It follows that $g(\theta, o_{X,Y}(\theta)) = \max_{o \in O-Y:(\forall o' \in X) u(\theta,o) \geq u(\theta,o')} g(\theta, o)$ $\geq \max_{o \in X \cup Z:(\forall o' \in X \cup Z) u(\theta,o) \geq u(\theta,o')} g(\theta, o) = g(\theta, o_{X \cup Z}(\theta))$. Thus $v(X, Y) = \sum_{\theta \in \Theta} p(\theta) g(\theta, o_{X,Y}(\theta)) \geq \sum_{\theta \in \Theta} p(\theta) g(\theta, o_{X \cup Z}(\theta)) = v(X \cup Z)$. ■

The following theorem shows that conveniently, at a leaf node, where we have decided for every outcome whether it is in or out, the heuristic value coincides with the value of that outcome set.

Theorem 44 *For any $X \subseteq O, \theta \in \Theta$, we have $o_{X,O-X}(\theta) = o_X(\theta)$ and $v(X, O - X) = v(X)$.*

Proof: Immediate using $O - (O - X) = X$. ■

6.7.4 The algorithm

We are now ready to present the algorithm.

Basic structure

We first summarize the backbone of our algorithm. A node in our search space is defined by a set of outcomes that are definitely "in" (X)[14] and a set of outcomes that are definitely out (Y). For a node at depth d, $X \cup Y$ always constitutes the first d outcomes for some fixed order of the outcomes; thus, a node has two children, one where the next outcome is added to X, and one where it is added to Y. The expansion order is fixed at putting the node in X first, and then in Y. The heuristic value (bound) of a node is given by $v(X, Y)$, as described above.

We can now simply apply A*; this, however, quickly fills up the available memory, so we resort to more space-efficient methods. We first present branch-and-bound depth-first search (expanding every node that still has a chance of leading to a better solution than the best one so far, in depth-first order) for our setting, and then IDA* (in which we maintain a target objective value, and do not expand nodes that do not have a chance of reaching this value; if we fail to find a solution, we decrease the target value and try again). (An overview of search methods such as these can be found in Russell and Norvig [2003].)

[14] We emphasize that this does *not* mean that the outcome will definitely be used by the mechanism corresponding to any descendant leaf node; rather, this outcome *may* be used by any descendant leaf node; and for any descendant leaf node, in the mechanism associated with this node, any type must receive an outcome at least as good to it as this one.

In the following, v is the heuristic for the current node. d is the depth of the current node. ω (a global variable) is the number of outcomes. CB (another global) is the outcome set corresponding to the best mechanism found so far. L (another global) is the expected value of the objective function for the best mechanism we have found so far. o_i is outcome i. The other variables are as described above.

```
BRANCH-AND-BOUND-DFS()
 CB := NULL
 L := −∞
 SEARCH1({}, {}, 0, 1)
 return CB

SEARCH1(X, Y, v, d)
 if d = ω + 1
  CB = X
  L = v
 else
  if v(X ∪ {o_d}, Y) > L
   SEARCH1(X ∪ {o_d}, Y, v(X ∪ {o_d}, Y), d + 1)
  if v(X, Y ∪ {o_d}) > L
   SEARCH1(X, Y ∪ {o_d}, v(X, Y ∪ {o_d}), d + 1)
```

Our implementation of IDA* is similar, except we do not initialize L to $-\infty$. Rather, we initialize it to some high value, and decrease it every time we fail to find a solution—either to a fraction of itself, or to the highest value that is still feasible (whichever is *less*). This also requires us to keep track of the highest value still feasible (given by HF, another global variable), so that we have to modify the search call slightly.

IDA*()
 CB := *NULL*
 L := *initial-limit*
 while CB = *NULL*
  $HF := -\infty$
  SEARCH2({}, {}, 0, 1)
  $L := \min\{HF,$ *fraction*$\cdot L\}$
 return CB

SEARCH2(X, Y, v, d)
 if $d = \omega + 1$
  $CB = X$
  $L = v$
 else
  if $v(X \cup \{o_d\}, Y) > L$
   SEARCH2($X \cup \{o_d\}, Y, v(X \cup \{o_d\}, Y), d + 1$)
  else if $v(X \cup \{o_d\}, Y) > HF$
   $HF := v(X \cup \{o_d\}, Y)$
  if $v(X, Y \cup \{o_d\}) > L$
   SEARCH2($X, Y \cup \{o_d\}, v(X, Y \cup \{o_d\}), d + 1$)
  else if $v(X, Y \cup \{o_d\}) > HF$
   $HF := v(X, Y \cup \{o_d\})$

Efficiently updating the heuristic

Rather than computing the heuristic anew each time, it can be computed much more quickly from information used for computing the heuristic at the parent node. For instance, when adding an outcome o to X, we will not have to change $o_{X,Y}(\theta)$ unless $u(\theta, o) > u(\theta, o_{X,Y}(\theta))$. As another example, when adding an outcome o to Y, we will not have to change $o_{X,Y}(\theta)$ unless $o_{X,Y}(\theta) = o$. In addition to this, maintaining appropriate data structures (such as a list of the outcomes sorted by objective value for a given type) allows us to quickly find the new outcome when we do need to make a change.

6.7.5 Individual rationality

We now show how to deal with an individual rationality constraint in this setting.

Theorem 45 *o_X is individually rational if and only if for every $\theta \in \Theta$, there is some $o \in X$ such that $u(\theta, o) \geq 0$.*

Proof: If for some $\theta \in \Theta$, there is no $o \in X$ such that $u(\theta, o) \geq 0$, o_X cannot give the agent nonnegative utility for type θ because o_X uses only outcomes from X; so it is not individually rational. On the other hand, if for every $\theta \in \Theta$, there is some $o \in X$ such that $u(\theta, o) \geq 0$, then o_X will give the agent nonnegative utility for that type θ, because o_X is constrained to choose an outcome that maximizes $u(\theta, \cdot)$ among outcomes from X, and at least one of the outcomes in X gives nonnegative utility. So it is individually rational. ■

It follows that when we have an individual rationality constraint, in our search procedures, we do not need to expand nodes where for some type θ, there are no outcomes left in $O - Y$ that give the agent a nonnegative utility for θ.

6.7.6 Experimental results

In this subsection, we compare the performances of branch-and-bound DFS and IDA* over our search space with the performance of the mixed integer programming approach described earlier (using CPLEX 8.0), on random instances drawn from three different distributions. In each case, we investigate both scalability in the number of types and in the number of outcomes.

Uniform distribution, no IR

For this distribution, each value $u(\theta, o)$ and each value $g(\theta, o)$ is independently and uniformly drawn from $[0, 100]$. No IR constraint applies (all utilities are nonnegative).

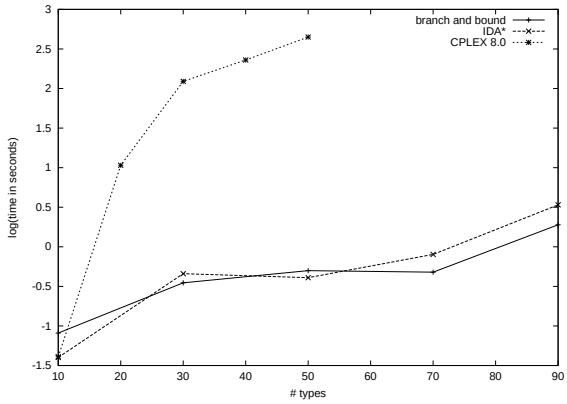

Figure 6.1: Performance vs. types for the uniform, no IR case with 20 outcomes.

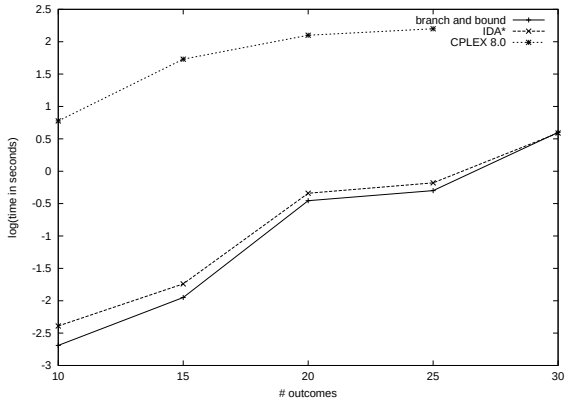

Figure 6.2: Performance vs. outcomes for the uniform, no IR case with 30 types.

Both versions of our algorithm outperform CPLEX soundly; our approach is especially more scalable in the number of types.

Uniform distribution, with IR

Now, each value $u(\theta, o)$ and each value $g(\theta, o)$ is independently and uniformly drawn from $[-50, 50]$. We apply an IR constraint (the agent can never get negative utility).

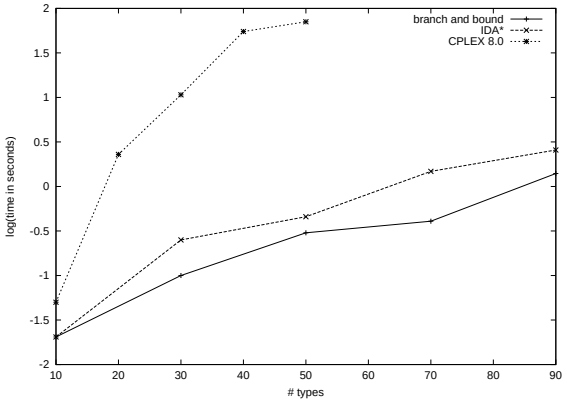

Figure 6.3: Performance vs. types for the uniform, with IR case with 20 outcomes.

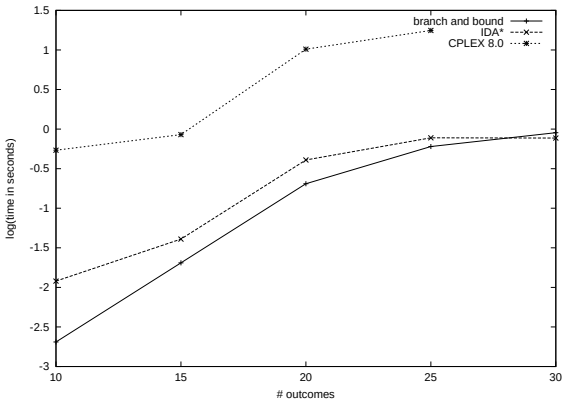

Figure 6.4: Performance vs. outcomes for the uniform, with IR case with 30 types.

Both versions of our algorithm still solidly outperform CPLEX, but the gaps are a little tighter; CPLEX manages to get a greater speedup factor out of the IR constraint.

Bartering

The final distribution corresponds to the bartering problem described earlier. The designer and the agent each have $m/2$ goods (for 2^m outcomes—each good can end up with either agent); the designer has a randomly drawn value (from $[0, 10]$) for each individual good (constituting g, which does not depend on θ in this case), and the agent has a randomly drawn value (from $[0, 10]$) for each individual good for each type (constituting u). The value of a bundle to an agent is the sum of the values of the individual goods.[15] If the total number of goods is odd, the agent gets one more good than the designer.

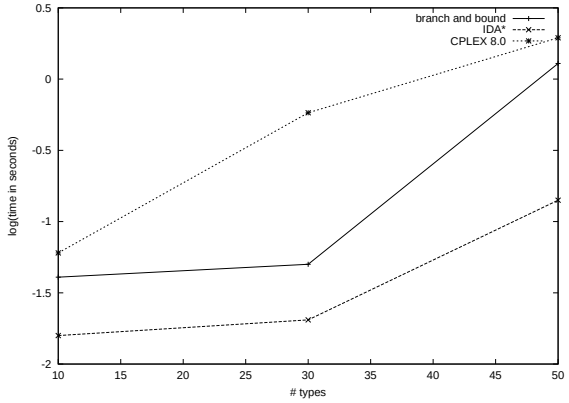

Figure 6.5: Performance vs. types for the bartering case with 32 outcomes.

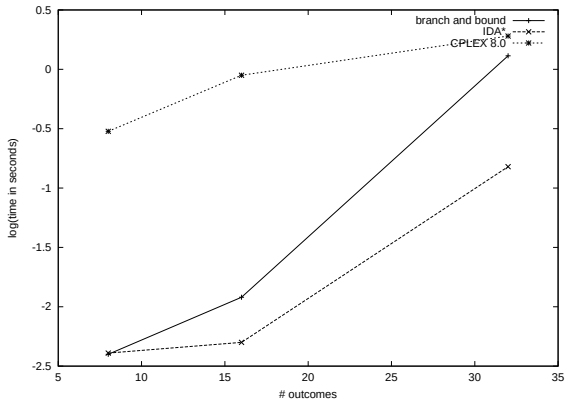

Figure 6.6: Performance vs. outcomes for the bartering case with 50 types.

The gaps here are much tighter, and it appears that CPLEX may in fact get the upper hand on even larger instances. (Space limitations prevented us from taking the experiments further.) CPLEX apparently makes very good use of the additional structure in this domain, whereas our algorithm

[15] There is nothing preventing our approach from having more complicated values over bundles; we simply felt it was nice to present the simplest example.

is not geared towards exploiting this structure. Also, IDA* seems to outperform branch-and-bound DFS now.

6.8 Structured outcomes and preferences

So far, we have only studied a flat representation of automated mechanism design problem instances, *e.g.* we assumed that all possible outcomes were explicitly listed in the input. However, in expressive preference aggregation, the outcome space is often too large to enumerate all the outcomes. Nevertheless, in such settings, the outcomes and the agents' preferences over them usually have some structure that allows the problem to still be concisely represented. In this section, we study one particular type of such structure: the agents may have to simultaneously decide on multiple, otherwise unrelated *issues*. In this case, the outcome space can be represented as the cross product of the outcome spaces for the individual issues. The next definition makes this precise.

Definition 30 *$O = O_1 \times O_2 \times \ldots \times O_r$ is a* valid decomposition *of O (where r is the number of* issues*) if the following two conditions hold:*

- *For each agent i, for each $1 \leq k \leq r$ there exists a function $u_i^k : \Theta_i \times O_k \rightarrow \mathbb{R}$ such that $u_i(\theta_i, (o^1, \ldots, o^r)) = \sum_{1 \leq k \leq r} u_i^k(\theta_i, o^k)$;*
- *For each $1 \leq k \leq r$ there exists a function $g^k : \Theta_1 \times \ldots \times \Theta_n \times O_k \rightarrow \mathbb{R}$ such that $g(\theta_1, \ldots, \theta_n, (o^1, \ldots, o^r)) = \sum_{1 \leq k \leq r} g^k(\theta_1, \ldots, \theta_n, o^k)$.*

We observe that when g is a social welfare function, the first condition implies the second, because if the first condition holds, $g(\theta_1, \ldots, \theta_n, (o^1, \ldots, o^r)) = \sum_{1 \leq i \leq n} u_i(\theta_i, (o^1, \ldots, o^r)) = \sum_{1 \leq i \leq n} \sum_{1 \leq k \leq r} u_i^k(\theta_i, o^k) = \sum_{1 \leq k \leq r} \sum_{1 \leq i \leq n} u_i^k(\theta_i, o^k)$, so that we can define $g^k(\theta_1, \ldots, \theta_n, o^k) = \sum_{1 \leq i \leq n} u_i^k(\theta_i, o^k)$.

We call automated mechanism design with a valid decomposition *multi-issue automated mechanism design*. It may seem that we can solve a multi-issue AMD instance simply by solving the AMD problem for each individual issue separately. However, doing so will in general not give the optimal mechanism. The reason is that in general, the designer may use one issue to tailor the incentives to get better results on another issue. For example, in an auction setting, one could think of the allocation as one issue, and the payments as another issue. Even when the designer is only concerned with bringing about the optimal allocation, the payments are still a useful instrument to give the bidders an incentive to bid truthfully. (We caution the reader that apart from this minor deviation, we do not consider the payments to be part of the outcome space O here.) As another example, we saw in Chapter 5 that using a single mechanism to decide on the donations to multiple charities can be more efficient than using a separate mechanism for each charity (Proposition 8). The hardness results later in this section will also imply that solving the AMD problem separately for each issue does not give the optimal solution. (The multi-issue AMD problem is NP-complete

even in settings where the corresponding single-issue AMD problem is in P, so if the approach of solving the problem separately for each issue were optimal, we would have shown that P=NP.)

6.8.1 Example: Multi-item auctions

Consider auctioning a set of distinguishable items. If each of the m items can be allocated to any of n agents (or to no agent at all), the outcome space O has size $(n+1)^m$ (one for each possible allocation). If, for every bidder, the bidder's valuation for any bundle of items is the sum of the bidder's valuations of the individual items in the bundle, then we can decompose the outcome space as $O = O_1 \times O_2 \times \ldots \times O_m$, where $O_k = \{0, 1, 2, \ldots, n\}$ is the set of all possible allocations for item k (0 indicating it goes to no agent). Agent i's utility function can be written as $u_i((o^1, o^2, \ldots, o^m)) = \sum_{k \in \{1,\ldots,m\}} u_i^k(o^k)$ where u_i^k is given by $u_i^k(i) = v_i(k)$ and $u_i^k(j) = 0$ for $j \neq i$, where $v_i(k)$ is agent i's valuation for item k.

Two extensions of this that also allow for decomposable outcome spaces are the following:

- An agent, if it does not receive an item, still cares which agent (if any) receives that item—that is, there are *externalities* (as discussed in Chapter 2, Section 2.4). Here we no longer always have $u_i^k(j) = 0$ for $j \neq i$. For example, John may prefer it if the museum wins the painting rather than a private collector, because in the former case he can still see the painting.

- Some items exhibit substitutability or complementarity (so an agent's valuation for a bundle is not the sum of its valuations of the individual items in the bundle), but the items can be partitioned into subsets so that there are no substitution or complementarity effects across subsets in the partition. In this case, we can still decompose the outcome space over these subsets. For example, a plane trip, a hotel stay, a cell phone and a pager are all for sale. The plane trip and the hotel stay are each worthless without the other: they are perfect complements. The cell phone and the pager each reduce the value of having the other: they are (partial) substitutes. But the value of the plane trip or the hotel stay has nothing to do with whether one also gets the cell phone or the pager. Thus, we decompose the outcome space into two issues, one indicating the winners of the plane trip and hotel stay, and one indicating the winners of the cell phone and the pager.

In each of these settings, the approach of this section can be used directly to maximize any objective that the designer has. (This requires that the vaulations lie in a finite interval and are discretized.)

6.8.2 Complexity

In this subsection we show that for the multi-issue representation, the three most important variants of the problem of designing a deterministic mechanism are NP-complete. Of course, the hardness results from Section 6.3 already imply this, because flatly represented problem instances are a special case of the multi-issue representation. However, it turns out that under the multi-issue representation, hardness occurs even in much more restricted settings (with small type spaces and a small outcome space for each issue).

Theorem 46 *The AMD problem for designing deterministic mechanisms without payments is NP-complete under the multi-issue representation, even when the objective is social welfare, there is only a single type-reporting agent (in addition to an agent that does not report a type), the probability distribution over Θ is uniform, there are only two possible types, and $|O_i| = 2$ for all i. (Membership in NP is guaranteed only if the number of agents is constant.)*

Proof: The problem is in NP because we can nondeterministically generate the full outcome selection function o (as long as the number of agents is constant, because otherwise there are exponentially many type vectors). To show NP-hardness, we reduce an arbitrary KNAPSACK instance (given by a set of pairs $\{(c_j, v_j)\}_{j\in\{1,\ldots,m\}}$, a cost limit C, and a value goal V) to the following single-agent deterministic multi-issue AMD instance, where payments are not allowed and the objective is social welfare. Let the number of issues be $r = m + 1$. For every $j \in \{1, \ldots, m+1\}$, we have $O_j = \{t, f\}$. The agent's type set, over which there is a uniform distribution, is $\Theta = \{\theta^1, \theta^2\}$, and the agent's utility function $u = \sum_{k\in\{1,\ldots,r\}} u^k$ is given by:

- For all $k \in \{1, \ldots, m\}$, $u^k(\theta^1, t) = AB$ where $A = 2 \sum_{j\in\{1,\ldots,m\}} c_j$ and $B = 2 \sum_{j\in\{1,\ldots,m\}} v_j$; and $u^k(\theta^1, f) = 0$.
- $u^{m+1}(\theta^1, t) = u^{m+1}(\theta^1, f) = 0$.
- For all $k \in \{1, \ldots, m\}$, $u^k(\theta^2, t) = c_k$, and $u^k(\theta^2, f) = 0$.
- $u^{m+1}(\theta^2, t) = C$, and $u^{m+1}(\theta^2, f) = 0$.

The part of the social welfare that does not correspond to any agent in the game is given by $u_0 = \sum_{k\in\{1,\ldots,r\}} u_0^k$ where

- For all $k \in \{1, \ldots, m\}$, $u_0^k(t) = 0$, and $u^k(f) = v_k A$.
- $u_0^{m+1}(t) = u_0^{m+1}(f) = 0$.

The goal social welfare is given by $G = \frac{A(mB+V)}{2}$. We show the two instances are equivalent. First suppose there is a solution to the KNAPSACK instance, that is, a subset S of $\{1, \ldots, m\}$ such that $\sum_{j\in S} c_j \leq C$ and $\sum_{j\in S} v_j \geq V$. Then consider the following mechanism:

- For all $k \in \{1, \ldots, m\}$, $o^k(\theta^1) = t$.
- For $k \in \{1, \ldots, m\}$, $o^k(\theta^2) = f$ if $k \in S$, $o^k(\theta^2) = t$ otherwise.
- $o^{m+1}(\theta^1) = f$, and $o^{m+1}(\theta^2) = t$.

First we show there is no incentive for the agent to misreport. If the agent has type θ^1, then it is getting the best possible outcome for each issue by reporting truthfully, so there is certainly no incentive to misreport. If the agent has type θ^2, reporting truthfully gives it a utility of $C + \sum_{j\in\{1,\ldots,m\},\notin S} c_j$, whereas reporting θ^1 instead gives it a utility of $\sum_{j\in\{1,\ldots,m\}} c_j$; so the marginal utility

of misreporting is $-C + \sum_{j \in S} c_j \leq -C + C = 0$. Hence there is no incentive to misreport. Now we show that the goal social welfare is reached. If the agent has type θ^1, the social welfare will be mAB. If it has type θ^2, it will be $\sum_{j \in S} v_j A + \sum_{j \in \{1,\ldots,m\}, \notin S} c_j + C \geq \sum_{j \in S} v_j A \geq VA$. Hence the expected social welfare is at least $\frac{mAB+VA}{2} = G$. So there is a solution to the AMD instance. Now suppose there is a solution to the AMD instance. If it were the case that, for some $j \in \{1, \ldots, m\}$, $o^j(\theta^1) = f$, then the maximum social welfare that could possibly be obtained (even if we did not worry about misreporting) would be $\frac{(m-1)AB+v_jA}{2} + \frac{\sum_{j \in \{1,\ldots,m\}} v_j A + C}{2} = \frac{(m-1)AB+\frac{AB}{2}+v_jA+C}{2} < \frac{mAB+VA}{2} = G$. Thus, for all $j \in \{1, \ldots, m\}$, $o^k(\theta^1) = t$. Now, let $S = \{j \in \{1, \ldots, m\} : o^j(\theta^2) = f\}$. Then, if the agent has type θ^2 and reports truthfully, it will get utility at most $C + \sum_{j \in \{1,\ldots,m\}, \notin S} c_j$, as opposed to the at least $\sum_{j \in \{1,\ldots,m\}} c_j$ that it could get for this type by reporting θ^1 instead. Because there is no incentive to misreport in the mechanism, it follows that $\sum_{j \in S} c_j \leq C$. Also, the total social welfare obtained by the mechanism is at most $\frac{mAB+\sum_{j \in S} v_j A+\sum_{j \in \{1,\ldots,m\}, \notin S} c_j+C}{2}$. Because $\sum_{j \in \{1,\ldots,m\}, \notin S} c_j + C < A$, and all the other terms in the numerator are some integer times A, it follows that this fraction is greater than or equal to the goal $\frac{mAB+VA}{2}$ (where the numerator is also an integer times A) if and only if $\sum_{j \in S} v_j \geq V$—and this must be the case because by assumption, the mechanism is a solution to the AMD instance. It follows that S is a solution to the KNAPSACK instance. ■

Theorem 47 *The AMD problem for designing deterministic mechanisms with payments is NP-complete under the multi-issue representation, even when the objective does not depend on the payments made, there is only a single type-reporting agent, the probability distribution over Θ is uniform, there are only two possible types, and $|O_i| = 2$ for all i. (Membership in NP is guaranteed only if the number of agents is constant.)*

Proof: It is easy to see that the problem is in NP. (We can nondeterministically generate the outcome function as before, after which setting the payments is a linear programming problem and can hence be done in polynomial time—presuming, again, that the number of agents is constant.) To show NP-hardness, we reduce an arbitrary KNAPSACK instance (given by a set of pairs $\{(c_j, v_j)\}_{j \in \{1,\ldots,m\}}$, a cost limit C, and a value goal V) to the following single-agent deterministic multi-issue AMD instance, where payments are allowed. Let the number of issues be $r = m + 1$. For every $j \in \{1, \ldots, m + 1\}$, we have $O_j = \{t, f\}$. The agent's type set, over which there is a uniform distribution, is $\Theta = \{\theta^1, \theta^2\}$, and the agent's utility function $u = \sum_{k \in \{1,\ldots,r\}} u^k$ is given by:

- For all $k \in \{1, \ldots, m\}$, $u^k(\theta^1, t) = c_k$, and $u^k(\theta^1, f) = 0$.
- $u^{m+1}(\theta^1, t) = C$, and $u^{m+1}(\theta^1, f) = 0$.
- For all $k \in \{1, \ldots, m\}$, $u^k(\theta^2, t) = 0$, $u^k(\theta^2, f) = c_k$.

- $u^{m+1}(\theta^2, t) = 0$, and $u^{m+1}(\theta^2, f) = C$.

The objective function $g = \sum_{k \in \{1,\ldots,r\}} g^k$ is given by

- For all $k \in \{1, \ldots, m\}$, $g^k(\theta^1, t) = 0$, and $g^k(\theta^1, f) = A$ where $A = 2 \sum_{j \in \{1,\ldots,m\}} v_j$.
- For all $k \in \{1, \ldots, m\}$, $g^k(\theta^2, t) = v_k$, $g^k(\theta^2, f) = 0$.
- $g^{m+1}(\theta^1, t) = g^{m+1}(\theta^1, f) = g^{m+1}(\theta^2, t) = g^{m+1}(\theta^2, f) = 0$.

The goal for the objective function is given by $G = \frac{mA+V}{2}$. We show the two instances are equivalent. We first observe a useful fact about the utility function: when there are no payments, for any outcome function, the incentive for the agent to misreport when it has type θ^1 is the same as the incentive for the agent to misreport when it has type θ^2. That is, for any outcome function o, $u(\theta^1, o(\theta^2)) - u(\theta^1, o(\theta^1)) = u(\theta^2, o(\theta^1)) - u(\theta^2, o(\theta^2))$. To see why, first consider that if we (say) set $o^k(\theta^1) = o^k(\theta^2) = f$ everywhere, obviously this is true. Then, whenever, for some k, we "flip" $o^k(\theta^1)$ to t, the second term (including the minus sign) on the left hand side decreases by the same amount as the first term on the right hand side. Similarly, whenever we "flip" $o^k(\theta^2)$ to t, the first term on the left hand side increases by the same amount as the second term (including the minus sign) on the right hand side. A corollary of this observation is that for this example, *payments cannot help us make the mechanism truthful*. For, if without payments, the mechanism would not be truthful, the agent would have an incentive to lie for both types (without payments). Then, if the agent needs to pay more for reporting one type than for the other, the agent will still have an (even bigger) incentive to lie for at least that type. Thus, we may as well assume payments are not possible. Now, suppose there is a solution to the KNAPSACK instance, that is, a subset S of $\{1, \ldots, m\}$ such that $\sum_{j \in S} c_j \leq C$ and $\sum_{j \in S} v_j \geq V$. Then consider the following mechanism:

- For all $k \in \{1, \ldots, m\}$, $o^k(\theta^1) = f$.
- For $k \in \{1, \ldots, m\}$, $o^k(\theta^2) = t$ if $k \in S$, $o^k(\theta^2) = f$ otherwise.
- $o^{m+1}(\theta^1) = t$, and $o^{m+1}(\theta^2) = f$.

(The payment function is 0 everywhere.) First we show there is no incentive for the agent to misreport. Because we observed that the incentive to misreport is the same for both types, we only need to show this for one type. We will show it when the agent's true type is θ^1. In this case, reporting truthfully gives utility C, and reporting θ^2 gives utility $\sum_{j \in S} c_j \leq C$. Hence there is no incentive to misreport. Now we show that the goal value of the objective is reached. If the agent has type θ^1, the value of the objective function will be mA. If it has type θ^2, it will be $\sum_{j \in S} v_j \geq V$. Hence the expected value of the objective function is at least $\frac{mA+V}{2} = G$. So there is a solution to the AMD instance. Finally, suppose there is a solution to the AMD instance. As a reminder, payments cannot help us, so we may assume they are always 0. If it were the case that, for some $j \in \{1, \ldots, m\}$, $o^k(\theta^1) = t$, then the maximum social welfare that could possibly be obtained (even if we did not

worry about misreporting) would be $\frac{(m-1)A+\sum_{j\in\{1,\ldots,m\}} v_j}{2} < \frac{mA}{2} < G$. Thus, for all $j \in \{1,\ldots,m\}$, $o^k(\theta^1) = f$. Now, let $S = \{j \in \{1,\ldots,m\} : o^j(\theta^2) = t\}$. The incentive for the agent to misreport when it has type θ^1 is then at least $-C + \sum_{j\in\{1,\ldots,m\}} c_j$, which should be less than or equal to 0, so that $\sum_{j\in\{1,\ldots,m\}} c_j \leq C$. Additionally, the expected value of the objective function is $\frac{mA+\sum_{j\in S} v_j}{2}$, which should be at least $G = \frac{mA+V}{2}$. It follows that $\sum_{j\in S} v_j \geq V$. Thus S is a solution to the KNAPSACK instance. ■

Theorem 48 *The AMD problem for designing deterministic mechanisms is NP-complete under the multi-issue representation, even when the objective is to maximize total payments made (under an IR constraint), there is only a single type-reporting agent, the probability distribution over Θ is uniform, there are only two possible types, and $|O_i| = 2$ for all i. (Membership in NP is guaranteed only if the number of agents is constant.)*

Proof: It is easy to see that the problem is in NP. (We can nondeterministically guess an outcome function, after which setting the payments is a linear programming problem and can hence be done in polynomial time.) To show NP-hardness, we reduce an arbitrary KNAPSACK instance (given by a set of pairs $\{(c_j, v_j)\}_{j\in\{1,\ldots,m+1\}}$, a cost limit C, and a value goal V) to the following single-agent deterministic multi-issue AMD instance, where we seek to maximize the expected payments from the agent. Let the number of issues be $r = m+1$. For every $j \in \{1,\ldots,m\}$, we have $O_j = \{t, f\}$. The agent's type set, over which there is a uniform distribution, is $\Theta = \{\theta^1, \theta^2\}$, and the agent's utility function $u = \sum_{k\in\{1,\ldots,r\}} u^k$ is given by:

- For all $k \in \{1,\ldots,m\}$, $u^k(\theta^1, t) = c_k A$ where $A = 4\sum_{j\in\{1,\ldots,m\}} v_j$; and $u^k(\theta^1, f) = 0$.
- $u^{m+1}(\theta^1, t) = 0$, and $u^{m+1}(\theta^1, f) = -CA$.
- For all $k \in \{1,\ldots,m\}$, $u^k(\theta^2, t) = v_k$, and $u^k(\theta^2, f) = 0$.
- $u^{m+1}(\theta^2, t) = 0$, and $u^{m+1}(\theta^2, f) = 0$.

The goal expected revenue is given by $G = \frac{AB+V}{2}$, where $B = \sum_{j\in\{1,\ldots,m\}} c_j$. We show the two instances are equivalent. First suppose there is a solution to the KNAPSACK instance, that is, a subset S of $\{1,\ldots,m\}$ such that $\sum_{j\in S} c_j \leq C$ and $\sum_{j\in S} v_j \geq V$. Then consider the following mechanism. Let the outcome function be

- For all $k \in \{1,\ldots,m\}$, $o^k(\theta^1) = t$.
- For $k \in \{1,\ldots,m\}$, $o^k(\theta^2) = t$ if $k \in S$, $o^k(\theta^2) = f$ otherwise.
- $o^{m+1}(\theta^1) = t,\ o^{m+1}(\theta^2) = f$.

Let the payment function be $\pi(\theta^1) = AB$, $\pi(\theta^2) = \sum_{j \in S} v_j$. First, to see that the IR constraint is satisfied, observe that for each type, the mechanism extracts exactly the agent's utility obtained from the outcome function. Second, we show there is no incentive for the agent to misreport. If the agent has type θ^1, reporting θ^2 instead gives a utility (including payments) of $-CA + \sum_{j \in S} c_j A - \sum_{j \in S} v_j \leq -CA + CA - \sum_{j \in S} v_j < 0$, which is what the agent would have got for reporting truthfully. If the agent has type θ^2, reporting θ^1 instead gives a utility (including payments) of $\frac{A}{4} - AB < 0$, which is what the agent would have got for reporting truthfully. Hence there is no incentive to misreport. Third, the goal expected payment is reached because $\frac{AB + \sum_{j \in S} v_j}{2} \geq \frac{AB+V}{2} = G$. So there is a solution to the AMD instance. Now suppose there is a solution to the AMD instance. The maximum utility that the agent can get from the outcome function if it has type θ^2 is $\frac{A}{4}$, and by the IR constraint this is the maximum payment we may extract from the agent when the reported type is θ^2. Because the goal is greater than $\frac{AB}{2}$, it follows that the payment the mechanism should extract from the agent when the reported type is θ^1 is at least $AB - \frac{A}{4}$. Because the maximum utility the agent can derive from the outcome function in this case is AB, it follows that the agent's utility (including payments) for type θ^1 can be at most $\frac{A}{4}$. Now consider the set $S = \{j \in \{1, \ldots, m\} : o^j(\theta^2) = t\}$. Then, if the agent falsely reports type θ^2 when the true type is θ^1, the utility of doing so (including payments) is at least $\sum_{j \in S} c_j A - CA - \frac{A}{4}$. This is to be at most the agent's utility for reporting truthfully in this case, which is at most $\frac{A}{4}$. It follows that $\sum_{j \in S} c_j A - CA - \frac{A}{4} \leq \frac{A}{4}$, which is equivalent to $\sum_{j \in S} c_j \leq C + \frac{1}{2}$. Because the c_j and C are integers, this implies $\sum_{j \in S} c_j \leq C$. Finally, because we need to extract at least a payment of V from the agent when type θ^2 is reported, but the utility that the agent gets from the outcome function in this case is at most $\sum_{j \in S} v_j$ and we can extract at most this by the IR constraint, it follows that $\sum_{j \in S} v_j \geq V$. Thus, S is a solution to the KNAPSACK instance. ■

The NP-hardness of automatically designing optimal deterministic mechanisms under the multi-issue representation was already implied by similar results for the unstructured (single-issue) representation. However, the fact that (unlike under the unstructured representation) NP-hardness occurs even with very small type sets is perhaps discouraging. On the other hand, one can be positive about the fact that the problem remains in NP (if the number of agents is constant), even though the representation is exponentially more concise. In the next subsection, we show that *pseudopolynomial-time* algorithms do exist for this problem (given a constant number of types). More significantly, in the subsection after that, we show that optimal *randomized* mechanisms can still be designed in polynomial time even under the multi-issue representation. Hence, it seems that this representation is especially useful when we allow for randomized mechanisms.

6.8.3 A pseudopolynomial-time algorithm for a single agent

In this subsection we develop a pseudopolynomial-time algorithm that shows that the first two multi-issue AMD problems discussed in the previous subsection are only *weakly* NP-complete. (A problem is only weakly NP-complete if it is NP-complete, but there exists an algorithm that would run in polynomial time if the numbers in the instance were given in unary, rather than binary—a *pseudopolynomial-time* algorithm.) This algorithm only works when there is only one type-reporting agent. While this is still a significant problem because of the conflict of interest between the designer and the agent, it is an interesting open problem to see if the algorithm can be generalized to settings with multiple agents.

Theorem 49 *If there is only a single agent, the number of types is a constant, and the objective does not depend on payments made, then the optimal deterministic mechanism can be found in pseudopolynomial time under the multi-issue representation using dynamic programming, both with and without payments, both for* ex post *and* ex interim *IR, and both for implementation in dominant strategies and for implementation in Bayes-Nash equilibrium.*

Proof: The dynamic program adds in the issues one at a time. For each $k \in \{0, 1, \ldots, r\}$, it builds a matrix which concerns a reduced version of the problem instance where only the first k issues are included. Let $r(\theta^i, \theta^j) = u(\theta^i, o(\theta^j)) - u(\theta^i, o(\theta^i))$, that is, the regret that the agent has for reporting its true type θ^i rather than submitting the false report θ^j. (These regrets may be negative.) Any outcome function mapping the reported types to outcomes defines a vector of $|\Theta|(|\Theta|-1)$ such regrets, one for each pair (θ^i, θ^j). Then, our matrix for the first k issues contains, for each possible regret vector v, a number indicating the highest expected value of the objective function that can be obtained with an outcome function over the first k issues whose regret vector is dominated by v. (One vector is said to be dominated by another if all entries of the former are less than or equal to the corresponding ones of the latter.) This entry is denoted $M^k[v]$. We observe that if v_1 dominates v_2, then $M^k[v_1] \geq M^k[v_2]$. If the absolute value of the regret between any two types is bounded by R, it suffices to consider $(2R+1)^{|\Theta|(|\Theta|-1)}$ regret vectors (each entry taking on values in $\{-R, -R+1, \ldots, 0, \ldots, R-1, R\}$). The matrix for $k = 0$ (i.e., when no issues have yet been added) is 0 everywhere. We then successively build up the next matrix as follows. When we add in issue k, there are $|O^k|^{|\Theta|}$ possibilities for setting the outcome function o^k from types to elements of O^k. Denoting a possible setting of o^k by a vector $w = (o^k(\theta^1), o^k(\theta^2), \ldots, o^k(\theta^{|\Theta|}))$, letting $g^k(w) = \sum_{\theta \in \Theta} g^k(\theta, o^k(\theta))$ be the total value gained in the objective function as a result of this vector, and letting $r(w) = (u^k(\theta^i, o^k(\theta^j)) - u^k(\theta^i, o^k(\theta^i))_{\{\theta^i \neq \theta^j\}}$ be the regret vector over this issue alone, we have the following recursive identity for $k > 0$: $M^k[v] = \max_w \{g^k(w) + M^{k-1}[v - r(w)]\}$. It is possible that, when we use this identity to fill in the matrices, the identity refers to an entry "outside" the previous matrix, that is, one of the entries of $v - r(w)$ has absolute value greater than R. If this occurs, one of the following two cases applies:

- One of the entries is greater than R. This means that the regret allowed for one of the pairs (θ^i, θ^j) is greater than the maximum it could be. We may reduce the value of this entry to R, without causing a reduction in the highest value of the objective function that can be obtained.

- One of the entries is smaller than $-R$. This means that the regret allowed for one of the pairs (θ^i, θ^j) is smaller than the minimum it could be. Hence, it is impossible to construct an outcome function that satisfies this, and hence we simply say $M^{k-1}[v - r(w)] = -\infty$.

Once we have constructed M^r, we can use this matrix to solve any of our deterministic automated mechanism design problems. If payments are not allowed, we simply look at the entry $M^r[(0, 0, \ldots, 0)]$, because this is the highest possible expected value of the objective function that we can obtain without the agent having positive regret anywhere. If payments are allowed, then we look at all the entries $M^r[v]$ where the regret vector v is such that we can set the payments so as to make every regret disappear—that is, where we can set π_θ such that for any θ^i, θ^j, we have $r(\theta^i, \theta^j) + \pi(\theta^j) - \pi(\theta^i) \leq 0$. (This is a simple linear program and can hence be solved in polynomial time.) Of all these entries, we choose the one with the highest value of the objective function. If we want to know not only the highest possible expected value of the objective function, but also a mechanism that achieves it, we need to store at each step not only the highest possible expected value for each matrix entry, but also a partial outcome function that achieves it. ■

6.8.4 A polynomial-time algorithm for randomized mechanisms

When we allow randomization in the mechanism, it turns out that an optimal mechanism can be designed in time that is polynomial in the length of the concise representation, as in the case of flatly represented instances (Section 6.4).

Theorem 50 *With a constant number of agents, the optimal randomized mechanism can be found in polynomial time under the multi-issue representation using linear programming, both with and without payments, both for* ex post *and* ex interim *IR, and both for implementation in dominant strategies and for implementation in Bayes-Nash equilibrium.*

Proof: We cannot simply use the linear program from Section 6.4, because it would have an exponential number of variables under the multi-issue representation. However, we can reduce the number of variables to a polynomial number. To this end, we observe:

- For all i, $E(u_i|(\hat{\theta_1}, \ldots, \hat{\theta_n}), \theta_i) = \sum_{(o^1,\ldots,o^r)\in O} P((o^1, \ldots, o^r)|(\hat{\theta_1}, \ldots, \hat{\theta_n})) \sum_{1\leq k\leq r} u_i^k(\theta_i, o^k)$
  $= \sum_{1\leq k\leq r} \sum_{(o^1,\ldots,o^r)\in O} P((o^1, \ldots, o^r)|(\hat{\theta_1}, \ldots, \hat{\theta_n}))u_i^k(\theta_i, o^k) =$
  $\sum_{1\leq k\leq r} \sum_{o_*^k\in O^k} u_i^k(\theta_i, o_*^k) \sum_{(o^1,\ldots,o^r):o^k=o_*^k} P((o^1, \ldots, o^r)|(\hat{\theta_1}, \ldots, \hat{\theta_n})) =$
  $\sum_{1\leq k\leq r} \sum_{o_*^k\in O^k} P(o^k = o_*^k|(\hat{\theta_1}, \ldots, \hat{\theta_n}))u_i^k(\theta_i, o_*^k)$.
- Similarly, $E(g|(\hat{\theta_1}, \ldots, \hat{\theta_n})) = \sum_{1\leq k\leq r} \sum_{o_*^k\in O^k} P(o^k = o_*^k|(\hat{\theta_1}, \ldots, \hat{\theta_n}))g^k((\hat{\theta_1}, \ldots, \hat{\theta_n}), o_*^k)$.

It follows that for the purposes at hand, we care only about the quantities $P(o^k = o_*^k|(\hat{\theta_1}, \ldots, \hat{\theta_n}))$, rather than about the entire distribution. There are precisely $\sum_{1\leq k\leq r} |O^k| \prod_{1\leq i\leq n} |\Theta^i|$ such probabilities, which is a polynomial number when the number of agents, n, is a constant. Additionally, only

$n \prod_{1 \leq i \leq n} |\Theta^i|$ variables are needed to represent the payments made by the agents in each case (or none if payments are not possible).

The linear program, which contains constraints for the IC notion and IR notion in question, and attempts to optimize some linear function of the expected value of the objective function and payments made, is now straightforward to construct. Because linear programs can be solved in polynomial time, and the number of variables and equations in our program is polynomial for any constant number of agents, the problem is in P. ■

6.9 Summary

In this chapter, we introduced *automated mechanism design*, which consists of solving for the optimal mechanism for the instance at hand using constrained optimization techniques. We showed that automatically designing optimal deterministic mechanisms is NP-hard in most cases, but designing the optimal randomized mechanism can be done in polynomial time using linear programming. Moreover, by requiring the probability variables in these linear programs to take on integer variables, we obtain a mixed integer programming approach for designing optimal deterministic mechanisms. We showed some initial applications, including divorce settlement, optimally auctioning one or more items, and deciding on whether to build public goods. We presented scalability results for the mixed integer/linear programming approaches; we also gave a special-purpose algorithm for a special case that outperforms the mixed integer programming approach. Finally, we studied a representation for instances of the automated mechanism design problem that is concise when there are multiple unrelated issues, and studied how this changes the complexity of the problem.

In the next few chapters, we will take a break from automated mechanism design and instead focus on the effects of agents' computational limitations on their behavior in (manually designed) mechanisms. We will return to the topic of designing mechanisms automatically in Chapter 10, where we automatically design mechanisms for agents with computational limitations.