Applications of Automated Mechanism Design

Vincent Conitzer
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Tuomas Sandholm
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Mechanism design is the art of designing the rules of the game so that desirable systemwide outcomes are obtained even though every agent in the system acts based on self-interest. Mechanism design has traditionally been a manual process. At UAI-02, we introduced *automated* mechanism design (AMD) [5]. In that paper and in other work, we studied its worst-case complexity [5, 6, 7]. This paper contains (to our knowledge) the first experimental work on AMD. We describe our implementation of AMD using a mixed integer/linear programming package (CPLEX 8.0), which we applied to a variety of scenarios that arise in different real-world settings. It created new optimal mechanisms for (divorce) dispute settlement, reinvented the Myerson optimal auction, invented optimal combinatorial auctions (a well-known important open research problem), and created new optimal mechanisms for public goods problems (both single-good and multi-good problems). We contrast the generated optimal mechanisms to the available mechanisms in the game theory literature. Finally, we present experimental scalability results of our implementation of AMD.

1 Introduction

Mechanism design is the art of designing the rules of the game so that a desirable outcome is reached even though the agents in the game behave selfishly. This is a difficult problem because the designer is uncertain about the agents' preferences and the agents may lie about their preferences. Mechanism design is receiving growing attention in the UAI community (e.g., [5, 15]).

Traditionally, the focus in mechanism design has been on designing mechanisms that are appropriate for a range of settings. While this approach has produced a number of famous mechanisms (for example, the VCG mechanism [16, 4, 10], the dAGVA mechanism [8, 2], and the Myerson optimal auction [13]), much of the space of possible settings is still left uncovered. In many (arguably most) cases where a mechanism is needed, the classical mechanisms are not satisfactory because they make assumptions on what the agents can do (for example, side payments are required for the mechanism to work); they pursue the wrong objective (for example, social welfare is pursued instead of maximal revenue); or they do not make use of all available information (such as probability distributions over the agents' preferences, or *types*), at the cost of the objective.

In contrast, in *automated mechanism design (AMD)* [5, 6, 7], a mechanism is *computed* on the fly for the setting at hand—a universally applicable approach. Our previous work on automated mechanism design studied the worst-case complexity of some versions of the abstract problem. In this paper, we describe the results of our implementation of an algorithm for automated mechanism design. Apart from the scalability results we provide towards the end of the paper, the paper's focus is on showing a variety of settings where we applied the AMD approach, and the results from doing so. We looked at some settings for which classical mechanisms are available (and are optimal). In these, the automated mechanism design approach rederived the classical mechanisms. We also looked at some settings for which (to our knowledge) no classical mechanisms are available (or classical mechanisms are available but nonoptimal—for example, a VCG mechanism can be used in an auction in which the auctioneer is actually seeking revenues), including some well-known open research areas. In these, the automated design approach was successful in deriving novel optimal mechanisms.

2 Automated mechanism design

We now formalize the automated mechanism design setting (in the same way as in our previous work [5, 6, 7]).

Definition 1 *In an* automated mechanism design setting, *we are given: 1. a finite set of outcomes O (in some cases, there may also be a cost function over these indicating how much needs to be paid for each outcome, e.g. in a public goods problem); 2. a finite set of N agents; 3. for each agent i, A. a finite set of types Θ_i, B. a probability distribution γ_i over Θ_i (in the case of correlated types, there is a single joint distribution γ over $\Theta_1 \times \ldots \times \Theta_N$), C. a utility function $u_i : \Theta_i \times O \rightarrow \mathbb{R}$;*[1] *4. an objective function whose expectation the designer wishes to maximize; 5. a specification of what tools are available to the designer (e.g., are payments possible, is randomization possible).*

There are many possible objective functions the designer might have. In this paper, we study the two best-known types of objective function: social welfare (both with and without taking payments into account), where the designer attempts to maximize the expected sum of the agents' utilities; and payment maximization, where the designer attempts to maximize the expected (net) sum of payments made to the center. For social welfare maximization, we study both settings where payments are possible, and where they are not possible. (In the former case, we assume *quasi-linear preferences*—every agent's utility is linear in the net payment that agent makes.)

The mechanism designer has to construct a game for the agents to play; how this game is played will determine the outcome chosen. (Additionally, it determines any side payments.) In designing the game, the mechanism designer seeks to maximize the expected value of the objective, under the assumption that the agents will play the game strategically. A useful result called the *revelation principle* states that the mechanism designer can restrict his attention to *direct revelation mechanisms*, where the agents report their types directly and where they never have any incentive to report them falsely. Thus, a *deterministic* mechanism is given by a function from reported type vectors to outcomes (and possibly to payment vectors). A *randomized* mechanism is given by a function from reported type vectors to probability distributions over outcomes (and possibly to payment vectors—but the agents will only care about the expected payment they have to make as long as they are risk-neutral.)

Furthermore, we need a definition of a *truthful* mechanism (one in which agents do not have incentives to lie about their types). The two best-known such definitions, and the ones studied in this paper, are the following. In *implementation in dominant strategies (DS)*, an agent never has an incentive to misreport her type even if she knows what all the other agents reported. In *implementation in Bayes-Nash equilibrium (BNE)*, an agent never has an incentive to misreport her type presuming that she knows nothing more about the other agents' types than the commonly known prior, and presuming that all the other agents will report truthfully. (We omit formal definitions because of space constraint.)

Finally, in many cases, the designer needs to make sure that the agents do not incur a loss as a result of participating in the mechanism (because the agent may then choose not to participate). This is known as an *individual rationality (IR)* constraint. We study *ex interim* IR, where it always makes sense for the agent to participate if she knows her own type but only the priors for the other agents; and *ex post* IR, where it always makes sense for the agent to participate even if she knows everyone's type.

3 Our implementation

As observed in our previous work on automated mechanism design [5, 6, 7], the problem of designing a randomized mechanism can typically be phrased as a linear program. Similarly, the problem of designing a deterministic mechanism can typically be phrased as a mixed integer program. (We do not describe these programs here because of the space constraint.) Our algorithm generates these mixed integer/linear programs, and subsequently uses a package (CPLEX 8.0) to solve them.

4 Divorce settlement

The first application setting in which we apply AMD is divorce settlement. We show several variants of the mechanism design problem, and the optimal solutions (mechanisms) to those variants generated by our AMD implementation. We first study a benevolent arbitrator, then a benevolent arbitrator that uses side 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.

4.1 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 between: (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

[1] Though this follows standard game theory notation [11], the fact that the agent has both a utility function and a type is perhaps confusing. The types encode the various possible preferences that the agent may turn out to have, and the agent's type is not known to the aggregator. The utility function is common knowledge, but because the agent's type is a parameter in the agent's utility function, the aggregator cannot know what the agent's utility is without knowing the agent's type. (In general, the type may also include knowledge about the other agents, but this can quickly lead to enormous type spaces. In this paper types will only indicate the agent's own preferences, as is common in much of the literature.)

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
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 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.

4.2 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.[2] 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.

4.3 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 being

[2] Classical mechanism design often separates the payments made from the social welfare calculation, allowing for easier analysis; one of the benefits of automated mechanism design is that the payments made can be easily integrated into the social welfare calculation in designing the mechanisms.

burned 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; we do not give them here because of space constraint.

5 Optimal auctions

In this section we show how AMD can be used to design auctions that maximize the seller's expected revenue (these are called *optimal* auctions). In many—if not most—auction settings, the seller would like to design the rules of the auction to accomplish this. 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 quasilinear preferences, *ex post* efficiency and IR can be accomplished in dominant strategies using the *Vickrey-Clarke-Groves (VCG) mechanism* [16, 4, 10]).

We first study auctioning off a single good, and show that AMD reinvents a known ladmark optimal auction mechanism for that setting. 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.

5.1 An optimal 2-bidder, 1-item auction

In this section, we 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 burn it (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, burn the item; otherwise, give the item to the highest bidder (a specific one of them in the case of a tie). This is exactly[3] the celebrated *Myerson auction* [13]. (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.)

[3]Apart from the payment rule generated, because CPLEX chooses to distribute the payments slightly differently across different type vectors.

5.2 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 [3, 17]. 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 [1].) We show that AMD can be used to generate optimal combinatorial auctions.

5.2.1 An optimal 2-bidder, 2-item combinatorial auction with complementarity

In our first combinatorial auction example, two items, A and B, are for sale. The auctioneer can award each item to any bidder, or burn it (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 burned. 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 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 burned.)

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 burning items, never from allocating items to a suboptimal bidder.

We omit the payment rule because of space constraint. The expected revenue from this 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 (in the number of bidders) optimal in multi-item auctions, that is, it maximizes revenue in the limit [12].[4] 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.)

5.2.2 An optimal 3-bidder, 2-item combinatorial auction with substitutability and complementarity

We omit this (large) example because of space constraint.

6 Public goods problems

Public goods problems are another key area of mechanism design [11]. In such problems, the agents have to make a joint decision that pertains to all of the agents. For example, the agents may vote over whether or not to build a bridge, but once the bridge is built, no agent can be excluded from using it. This gives rise to a *free-riding* problem. The *Groves mechanism* is a general solution to this problem (for agents with quasilinear preferences). It guarantees that the ex post social welfare maximizing choice is made, that the mechanism is *ex post* IR, and the truthful revelation of preferences is each agent's dominant strategy [10]. The Groves mechanism collects payments from the agents depending on what preferences they revealed; these payments set the correct incentives for the agents to tell the truth. Unfortunately, the Groves mechanism does not maintain budget balance. Usually the sum of payments is greater than the cost of the project, and these extra payments have to be burned (redistributing them back to the agents or to any cause that the agents care about would distort the incentives for truth-telling). In fact, for the general class of quasilinear preferences, there exists no mechanism that achieves budget balance, truth-dominance, and ex post efficiency (social welfare maximization) [9].

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. We use AMD to design a truth-dominant, ex post IR mechanism that is as ex post efficient as possible—*taking into account money burning as a loss in efficiency*.

6.1 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 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 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

[4] 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 [12].)

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, our mechanism achieves everything we hope for.

For Bayes-Nash implementation among agents with quesilinear preferences, the *dAGVA* mechanism achieves budget balance, truth-telling as the equilibrium strategy, and ex post efficiency [8, 2]. However, no mechanism achieves these properties and ex post IR for general (quasilinear) preferences [14]. As our mechanism above shows, AMD can circumvent this impossibility in specific settings.

6.2 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.

7 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 optimal combinatorial auction from subsection 5.2.2, 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 [5, 6, 7].)

#agents	D/DS	R/DS	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 (DS) 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/DS	R/DS	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/DS	R/DS	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/DS	R/DS	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 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/DS	R/DS	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.

8 Conclusions

In this paper, we presented (to our knowledge) the first applications of automated mechanism design (AMD). This yielded several mechanisms for a divorce settlement scenario (each of them optimal for a particular purpose); it rederived the Myerson optimal auction for selling a single good; it produced novel optimal combinatorial auction mechanisms for maximizing the seller's expected revenue (a recognized difficult problem in the combinatorial auction literature); and it produced novel optimal mechanisms for public goods problems (both with a single good and with multiple goods). As the examples show, AMD can circumvent seminal impossibility results in mechanism design because the mechanism is not designed for a general class of problems, but rather for the specific setting at hand. Even when the optimal mechanism—created using AMD—does not circumvent the impossibility, it always minimizes the pain entailed by impossibility.

Finally, we studied the scalability of AMD on unstructured problems. The experiments show that the runtime is heavily dependent upon 1. the number of agents (the more the harder), 2. the number of types per agent (the more the harder), 3. whether implementation in dominant strategies or in Bayes-Nash equilibrium is required (the former is harder), 4. whether randomization is allowed (deterministic mechanism design is harder), and 5. which objective is pursued (from social welfare with payments that are not taken into account (easy) to payment maximization (hard)). Other variables, such as the number of outcomes and which (if any) IR constraint is used, turned out to matter much less. In the case that some of these variables in the problem can be changed in order to achieve feasibility, these results are a valuable guideline in selecting which variables to change. On the other hand, given that in many settings, we will not be able to change these variables, these results also indicate where new, faster algorithms will be most valuable.

References

[1] Mark Armstrong. Optimal multi-object auctions. *Review of Economic Studies*, 67:455–481, 2000.

[2] Kenneth Arrow. The property rights doctrine and demand revelation under incomplete information. In M Boskin, editor, *Economics and human welfare*. New York Academic Press, 1979.

[3] Christopher Avery and Terrence Hendershott. Bundling and optimal auctions of multiple products. *Review of Economic Studies*, 67:483–497, 2000.

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

[5] Vincent Conitzer and Tuomas Sandholm. Complexity of mechanism design. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 103–110, Edmonton, Canada, 2002.

[6] Vincent Conitzer and Tuomas Sandholm. Automated mechanism design: Complexity results stemming from the single-agent setting. In *The 5th International Conference on Electronic Commerce (ICEC-03)*, pages 17–24, Pittsburgh, PA, USA, 2003.

[7] Vincent Conitzer and Tuomas Sandholm. Self-interested automated mechanism design and implications for optimal combinatorial auctions. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, New York, NY, 2004.

[8] C d'Aspremont and L A Gérard-Varet. Incentives and incomplete information. *Journal of Public Economics*, 11:25–45, 1979.

[9] J Green and J-J Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45:427–438, 1977.

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

[11] Andreu Mas-Colell, Michael Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[12] Dov Monderer and Moshe Tennenholtz. Asymptotically optimal multi-object auctions for risk-averse agents. Technical report, Faculty of Industrial Engineering and Management, Technion, Haifa, Israel, February 1999.

[13] Roger Myerson. Optimal auction design. *Mathematics of Operation Research*, 6:58–73, 1981.

[14] Roger Myerson and Mark Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 28:265–281, 1983.

[15] Ryan Porter, Amir Ronen, Yoav Shoham, and Moshe Tennenholtz. Fault tolerant mechanism design. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, Edmonton, Canada, 2002.

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

[17] Rakesh V. Vohra. Research problems in combinatorial auctions. Mimeo, version Oct. 29, 2001.