Automated Design of Multistage Mechanisms*

Tuomas Sandholm
Carnegie Mellon University
Computer Science Department
Pittsburgh, PA 15213, USA
sandholm@cs.cmu.edu

Vincent Conitzer
Carnegie Mellon University
Computer Science Department
Pittsburgh, PA 15213, USA
conitzer@cs.cmu.edu

Craig Boutilier
University of Toronto
Department of Computer Science
10 King's College Road
Toronto, Ontario, Canada M5S 3G4
cebly@cs.toronto.edu

Abstract

Mechanism design is the study of preference aggregation protocols that work well in the face of self-interested agents. We present the first general-purpose techniques for automatically designing multistage *mechanisms. These can reduce elicitation burden by only querying agents for information that is relevant given their answers to previous queries. We first show how to turn a given (e.g., automatically designed using constrained optimization techniques) single-stage mechanism into the most efficient corresponding multistage mechanism given a specified elicitation tree. We then present greedy and dynamic programming (DP) algorithms that will determine the elicitation tree (optimal in the DP case). Next, we show how the query savings inherent in the multistage model can be used to design the underlying single-stage mechanism to maximally take advantage of this approach. We illustrate all of these techniques on an optimal auction example. Finally, we present negative results on the design of multistage mechanisms that do not correspond to* dominant-strategy *single-stage mechanisms: an optimal multistage mechanism in general has to randomize over queries to hide information from the agents.*

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

1. Introduction

In multiagent settings, often an *outcome* (e.g., presidents, joint plans, allocations of resources) must be chosen based on the preferences of a set of agents. Since the preference aggregator generally does not know these preferences *a priori*, the agents must report their preferences to the aggregator. Unfortunately, an agent may have an incentive to misreport its preferences in order to mislead the aggregator into selecting an outcome that is more desirable to the agent than the outcome that would be selected if the agent revealed its preferences truthfully.

Mechanism design is concerned with the creation of preference aggregation rules that lead to good outcomes in spite of such strategic behavior by agents. Classical mechanism design provides some general mechanisms, which, under certain assumptions, satisfy some notion of nonmanipulability and maximize some objective. Such mechanisms do not rely on (even probabilistic) information about the agents' preferences (e.g., the Vickrey-Clarke-Groves (VCG) mechanism [24, 7, 14]), or can be easily applied to any distribution over preferences [13, 1, 18, 17]. However, these general algorithms only work in restricted settings (e.g., requiring the possibility of side payments), and may not reflect the designer's objectives.

Recently, *automated mechanism design (AMD)* has been proposed as a means to design mechanisms automatically for the setting at hand [8, 10, 12, 4]. This constrained-optimization based approach produces optimal special-purpose mechanisms even in settings for which no good general mechanisms are known, or for which an impossibility result precludes the existence of good general mechanisms for the class of instances (but not the existence of a good mechanism for the specific instance at hand). However, all prior work on general-purpose AMD has focused on single-stage mechanisms, in which all agents reveal their preferences completely and simultaneously. This is problematic for several reasons. First and foremost, agents may need to invest computational (or other) resources to determine their preferences over outcomes (e.g., when bidding on trucking tasks, an agent needs to solve, for each subset of tasks, a complex vehicle-routing problem). Second, the agents lose all privacy about their preferences. Third, it can require a large amount of communication. While the third reason applies only when the space of possible preferences is large, the first two reasons are significant even in very small examples.

Much of this computation, communication, and privacy loss is unnecessary when certain aspects of an agent's preferences have no influence on the final outcome. For instance, if a second agent can perform a task at much lower cost than a first, we need not determine precisely how suboptimal assigning the task to the first agent is. Unfortunately, single-stage mechanisms cannot take advantage of this: we cannot *a priori* rule out the need to know the first agent's precise preferences for the task—this only becomes apparent after receiving information from the second.

Our solution is to use *multistage* mechanisms, where the aggregator queries the agents about certain aspects of their preferences, and chooses the next query to ask (and who to ask it of) based on answers to earlier queries. In a non-game-theoretic setting, a move to multistage protocols can yield an exponential savings in bits communicated [15]. In mechanism design settings, such a move can yield an exponential savings in communication and the aggregator's computation [11]. In combinatorial auctions, the savings in communication can even be *super*-exponential [3].

Multistage mechanisms have been *manually* designed for several applications, such as voting [9], 1-object auctions (e.g. [6]), and combinatorial auctions (see reviews by [21, 19]). In this paper, we introduce the first general-purpose techniques for *automated* design of multistage mechanisms.[1] We adopt a specific approach: we first design a single-stage mechanism using existing techniques, and then convert it into a multistage mechanism in various ways. We also show how to design the underlying single-stage mechanism to maximally take advantage of this approach, so that the approach does not come at a loss.

[1]Previous work has studied the design of multistage mechanisms in a strategic multi-party computation setting [23, 2], but the issues in that setting are very different from those that we study in this paper. For example, the key strategic issue in that work is that an agent may be tempted not to invest the effort necessary to determine its private information.

2. The model

2.1. Automated design of single-stage mechanisms

In this subsection, we review the relevant definitions and results from the single-stage AMD literature. In a single-stage AMD setting, we are given: 1) a finite set of outcomes O;[2] 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 $\Theta_1 \times \ldots \times \Theta_N$), and (c) a utility function $u_i : \Theta_i \times O \rightarrow \mathbb{R}$;[3] 4) an objective function $g : \Theta_1 \times \ldots \times \Theta_N \times O \rightarrow \mathbb{R}$ whose expectation the designer wishes to maximize. Possible designer objectives are many (e.g., *social welfare*, or maximizing the sum of agent utilities for the chosen outcome).

By the *revelation principle* [16], we can restrict attention to *truthful*, *direct revelation mechanisms*, where agents report their types directly and never have an incentive to misreport them. In general, mechanisms may choose the outcome randomly. Thus, a mechanism consists of a distribution selection function $p : \Theta_1 \times \ldots \times \Theta_N \rightarrow \Delta(O)$, where $\Delta(O)$ is the set of probability distributions over O. A mechanism is a *dominant strategy mechanism* if truthtelling is optimal regardless of what other agents report. In other words, for any agent i, type vector $(\theta_1, \ldots, \theta_i, \ldots, \theta_N)$, and alternative report $\hat{\theta}_i \in \Theta_i$, we have $E_{o|\theta_1,..,\theta_i,..,\theta_n} u_i(\theta_i, o) \geq E_{o|\theta_1,..,\hat{\theta}_i,..,\theta_n} u_i(\theta_i, o)$. If telling the truth is optimal only *given* that the other agents are truthful, we have a *Bayes-Nash equilibrium (BNE)* mechanism. That is, in a BNE mechanism, for any i, $\theta_i \in \Theta_i$, and alternative report $\hat{\theta}_i \in \Theta_i$, we have $E_{(\theta_1,..,\theta_{i-1},\theta_{i+1},..,\theta_N)|\theta_i} E_{o|\theta_1,..,\theta_i,..,\theta_n} u_i(\theta_i, o) \geq E_{(\theta_1,..,\theta_{i-1},\theta_{i+1},..,\theta_N)|\theta_i} E_{o|\theta_1,..,\hat{\theta}_i,..,\theta_n} u_i(\theta_i, o)$.[4]

Given that the mechanism is allowed to choose the outcome at random, the problem of designing an optimal single-stage mechanism can be solved in polynomial time (given that the number of agents is constant) using linear programming [8]. The decision variables of that linear program are the following: for every type vector θ and every outcome o, there is a decision variable $p(\theta, o)$ that de-

[2]Payments to/from agents can be part of the outcome.

[3]The utility function is parameterized by type; while the u_i are common knowledge, the types encode (private) preferences [16]. The restriction to a finite type space is somewhat limiting. However, continuous spaces can be handled via suitable discretization of the type space. The discretization can be fixed in advance with an analysis of its impact on incentives and efficiency (as in recent research on limited revelation auctions [5]). Or, it may be optimized within the AMD model itself; this latter point is the subject of current research.

[4]In settings where participation is voluntary, the AMD formulation also includes participation (or *individual rationality*) constraints: no agent is worse off participating in the mechanism than not. Techniques for handling them in single-stage AMD can be applied to our multistage case without modification.

termines the probability of that outcome given that type vector. It is straightforward to check that the incentive compatibility constraints above, as well as the expectation of the objective, are linear functions of these variables, which gives us the linear program. Generating and solving this linear program is all that is required to have a basic approach to automatically designing single-stage mechanisms, and it is in fact the approach that we use in this paper to generate single-stage mechanisms.

2.2. Automated design of multistage mechanisms

In multistage AMD, the input includes—in addition to the input for single-stage AMD—a set of queries Q and a set of answers A.[5] Each query q is associated with a particular agent i (of whom q would be asked),[6] and the answer that the agent would give to q (when answering truthfully) is given by the function $a : Q \times \Theta_i \rightarrow A$, where $a(q, \theta_i)$ is i's truthful answer to query q when i's type is θ_i. This implies that there is only one truthful response to any $q \in Q$; thus, each query partitions the agent's type space. Upon receiving answer a to q from agent i, the mechanism can infer (assuming truthfulness) i's type is in $\{\theta_i \in \Theta_i : a(q, \theta_i) = a\}$.

A multistage mechanism M *corresponds* to a given single-stage mechanism S if, for each type vector θ reported by the agents, both M and S choose each outcome o with the same probability. Suppose M corresponds to some S where truth-telling is dominant. It is not hard to see that M has truthtelling as an *ex-post* equilibrium, regardless of the the results of previous queries revealed. That is, truthtelling is optimal (regardless of an agent's beliefs) whenever all other agents answer queries truthfully.[7] This implies that we never need to randomize over query choice (though this no longer holds if S is not a dominant-strategy mechanism; see Section 7).

For these reasons, apart from Section 7, we focus exclusively on multistage mechanisms that correspond to dominant-strategy single-stage mechanisms. Thus, we can restrict ourselves to mechanisms that select the next query deterministically based on answers to prior queries; moreover, we need not worry about incentives.

[5]Assuming a single answer set (rather than distinct A_q for each query q) comes w.l.o.g. One set of interest is $A = \{\text{yes, no}\}$.

[6]In this paper we will restrict our attention to the case where we query one agent at a time; however, our approach is easily extended to settings where we query multiple agents at the same time. We note, however, that querying agents one at a time leads to the largest possible savings in the number of queries.

[7]Ex post is weaker than dominant strategies, but stronger than BNE. Note that even if S is a dominant-strategy mechanism, M need not be: if an agent makes her answer dependent on the history of queries asked, another agent may have an incentive to lie about her type in order to influence which queries the former is asked.

Under these restrictions, a *multistage mechanism* is defined by: 1) a tree with nodes V and edges E; 2) for each internal (non-leaf) node v, an agent i and a query q to that agent; 3) a one-to-one correspondence between possible answers to the query at node v and children of node v; 4) for each node v and outcome o, a probability that, given that we reach v, we stop asking queries and choose outcome o. (In the case where v is a leaf, these probabilities must sum to one.) An *elicitation tree* is a multistage mechanism without outcome probabilities. We denote by I_v the *information set* at node v (i.e., the set of type vectors consistent with the answers that lead to v). We generally assume an elicitation tree is *complete*: I_l is a single type vector for any leaf l.[8]

We study several variants of multistage AMD. First, we may either start from a given single-stage mechanism (e.g., computed by single-stage AMD software) and turn it into a corresponding multistage mechanism, or we may impose no constraints on the single-stage version of the multistage mechanism. Second, we may either assume that the elicitation tree (hence the query order) is given beforehand, or we may impose no constraints on it.

3. A small example

In this section, we illustrate various notions for automatically designing multistage mechanisms using a single, simple example. Suppose a divorcing couple jointly owns a painting, and an arbitrator has to decide the fate of the painting. There are 5 options: (1) the husband keeps the painting; (2) the wife keeps it; (3) the painting remains jointly owned, but is hung in a museum; (4) it is cut into pieces which are given to the husband; and (5) it is cut up with pieces given to the wife. The husband and wife each have two possible types: type L is associated with relative indifference toward the painting, and type H with deep attachment. Each has type L with probability 0.8 and type H with probability 0.2. To maximize social welfare, the arbitrator would like to give the painting to whomever cares for it more; but since a party who cares little would prefer having it over not, the arbitrator must design appropriate incentives to ensure truthful reporting. The utility function for each party is the "same." Keeping the painting gives utility 2 (type L) or 100 (H). The other party getting the painting gives utility 0 (for either type). The museum outcome gives utility 1.5 (L) or 40 (H). Receiving pieces gives utility -9 while not even getting the pieces gives utility -10

[8]This does not imply that the mechanism will ask all queries and uniquely determine a type vector: the concrete outcome probabilities, specifically, the possibility of terminating at an interior node will typically preclude this.

(for either type).[9]

Our goal is to find a dominant-strategy (possibly randomized) mechanism (without payments) that maximizes expected social welfare. First we find the optimal single-stage mechanism. Solving this example using the methodology described in Subsection 2.1 yields the following randomized mechanism (the probabilities are rounded):

	wife L	wife H
husband L	Museum	0.96 Wife keeps; 0.04 Husband gets pieces
husband H	0.96 Husband keeps; 0.04 Wife gets pieces	0.47 Husband keeps; 0.40 Wife keeps; 0.13 Wife gets pieces

In spite of the symmetry between the husband and the wife, the mechanism is asymmetric. Of course, other optimal solutions exist (e.g., where the roles of husband and wife are interchanged).

Now we consider how to turn this single-stage mechanism into a corresponding multistage mechanism (i.e., with the same outcome probabilities). First, suppose the elicitation tree is given, with the wife's type elicited first. Fig. 1 shows the optimal multistage mechanism. (Why this is so will become apparent.) This mechanism saves one query with probability $0.2 \cdot 0.4 = 0.08$.

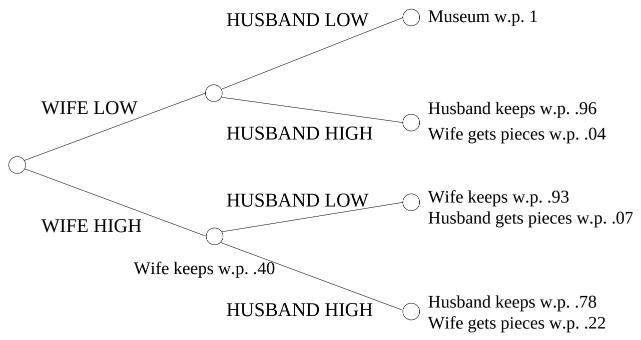

Figure 1. The optimal elicitation tree given the single-stage mechanism and given that the wife is queried first. When an internal node has a probability-outcome pair associated with it, we terminate early at that node with that probability, with that outcome; with the remaining probability, we move on to the next query.

If the elicitation tree (query order) is not fixed, the optimal mechanism is that given in Fig. 2. It turns out that greater savings can be obtained by eliciting the husband's type first: this mechanism saves a query with probability $0.2 \cdot (0.04 + 0.47) = 0.10$. (This is due to the asymmetry of the single-stage mechanism from which we are starting.)

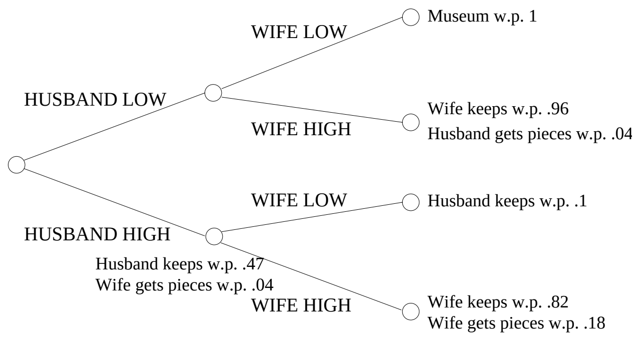

Figure 2. The optimal mechanism when asking the husband first.

Suppose queries to the husband are slightly more expensive than those to the wife, so that we would rather save on husband queries (unlike the previous mechanism). If we allow a different optimal single-stage mechanism, namely the analog of the one above with the husband and wife roles switched (which remains optimal due to the problem symmetry), then the optimal multistage mechanism that corresponds to this saves a query to the husband (rather than the wife) with probability 0.10, giving greater cost savings.

Finally, if we are willing to sacrifice optimality of the single-stage mechanism to obtain greater query savings, this may again change the mechanism. For example, if we make the cost of querying sufficiently large, it will be optimal to not ask any queries, and always choose the same outcome.

One interesting additional motivation for automatically designing multistage mechanisms is that the tree-based representation of a multistage mechanism may be *easier to understand* for a human than the tabular form of a single-stage mechanism—especially if the tree is relatively small.

4. Converting a single-stage mechanism into a multistage mechanism

In this section we develop methods for converting a given (e.g., automatically designed) single-stage mechanism into an equivalent multistage mechanism which saves on elicitation costs. In Subsection 4.1 we develop methods for the case where the elicitation tree (query order) is given. In Subsection 4.2 we generalize the approach to the case where the elicitation tree is not given, but can be chosen endogenously.

[9]This problem has some similarity to King Solomon's dilemma; however, when that dilemma is discussed in the economics literature [20], it is assumed that there is only one rightful mother, and both women know who it is—unlike our problem, where the agents do not know each others' types.

4.1. Given elicitation tree

We first solve the simplest of our problems: converting a single-stage mechanism into the most efficient multistage mechanism for a given elicitation tree. This problem can be motivated by considering exogenous constraints on query order (e.g., agents available at different times, or when the optimal ordering is readily available). More importantly, this setting serves as a stepping stone to more general techniques below. Our key technique is to "propagate up" probability from the leaves to internal nodes where this is possible.

Lemma 1 *Let multistage mechanism M correspond to single-stage mechanism S. Suppose that for some internal node v in the elicitation tree (with exit prob. e_v) and outcome o, all the leaves of the subtree T_v rooted at v assign a probability of at least $p > 0$ to outcome o. Then the following modification M' of M corresponds to S: (1) At node v, exit with o with probability $(1 - e_v)p$; (2) Subtract p from the probability assigned to o at each leaf of T_v; (3) Divide all the outcome probabilities at leaves T_v by $1 - p$.*

Proof: Consider the probability $p(\theta, o')$ that outcome o' will be selected given type vector θ in M'. If θ does not lead to v, clearly $p(\theta, o')$ is the same in M and M'; so assume that it does. If $o' = o$ (the outcome that we exit early with), then the probability of selecting o' at v is now the early-exit probability p, plus the probability that we do not exit early but choose outcome o' later, which is $(1-p)(p(\theta,o')^{old} - p)/(1-p) = p(\theta,o')^{old} - p$. Hence the total probability is $p(\theta,o')^{old}$; i.e., it did not change. If $o' \neq o$, then the probability of selecting o' at v is the probability that we do not exit early with o and choose outcome o' later, which is $(1-p)(p(\theta,o')^{old})/(1-p) = p(\theta,o')^{old}$. Hence for any θ, M and M' select o' with the same probability. ■

We note that the ability to propagate probability up in this manner even when the distributions at the leaves are not identical makes this different from the standard framework in communication complexity theory. (In addition, we may have a restricted query language, and we have a prior distribution over the inputs.)

If we propagate up as much probability as possible, we obtain the optimal mechanism (for a given S and tree):

Theorem 1 *Suppose we apply Lemma 1 until we can apply it no further (that is, until for any internal node v and outcome o, there is at least one leaf of the subtree rooted at v that assigns probability 0 to o). Then the resulting multistage mechanism saves the most queries (or, in the case of different query costs, the greatest query cost) among multistage mechanisms corresponding to the given single-stage mechanism and the given elicitation tree.*

Proof: Any mechanism with the same elicitation tree that saves more queries (or greater query cost) must, for some node v, have a greater probability of exiting early at or before v than the mechanism generated by applying Lemma 1. It follows that for at least one outcome o, the former mechanism has a probability of exiting early at or before v with this o that is greater than $\sum_{o \in O} \min_{\theta \in I_v} p(\theta, o)$. But then, there is some $\theta \in I_v$ such that $p(\theta, o)$ is smaller than the probability of exiting early at or before v with outcome o. So the mechanism does not correspond to the given single-stage mechanism. ■

As an example, we derive the mechanism of Fig. 2. We start from a mechanism that saves no queries (Fig. 3).

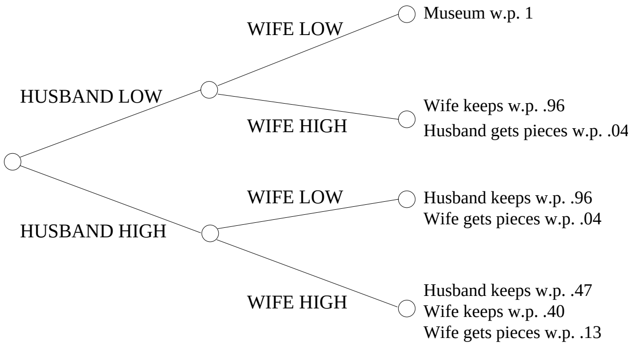

Figure 3. Multistage mechanism that saves no queries at all.

At the node after the husband reports "high", the husband keeps the painting with $p \geq .47$ in all subsequent leaves. So we can propagate this probability up (Fig. 4). At the same

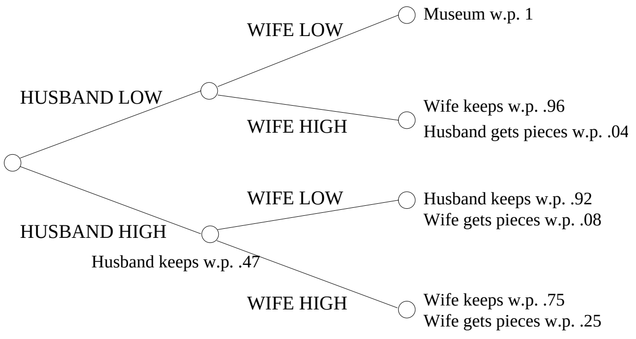

Figure 4. Some probability propagated up.

node, the wife gets the pieces of the painting with $p \geq .08$

in all subsequent leaves. Propagating this up results in the mechanism of Fig. 2.

The following corollary characterizes the probability of exiting early at or before a given node. This will be helpful in our use of the "propagating probabilities up" technique within all of the algorithms discussed later in the paper.

Corollary 1 *In a multistage mechanism that saves a maximum number of queries, for any type vector θ such that node v will be reached if the mechanism does not exit early, the probability that we will reach v and not exit early at v is $1 - \sum_{o \in O} \min_{\theta \in I_v} p(\theta, o)$. Hence, given that we have not exited early at or before node v, and we transition from node v to node w, the probability of exiting early at node w is $1 - (1 - \sum_{o \in O} \min_{\theta \in I_w} p(\theta, o))/(1 - \sum_{o \in O} \min_{\theta \in I_v} p(\theta, o))$.*

4.2. Endogenously determined elicitation tree

In this section we develop methods for converting a single-stage mechanism into a multistage one, *without constraints on the elicitation tree (query order)*. We first provide a greedy algorithm, and show two ways in which it can "fail" (i.e., yield an arbitrarily small fraction of the query savings available). We then give an optimal dynamic program.

4.2.1. Greedy algorithm Our greedy algorithm chooses the query at each stage so as to maximize the probability of being able to exit immediately after this query given the preceding queries and responses. Letting $U(I, q, a_q)$ denote the information state that results from being in information state I and then receiving answer a to query q, we define the algorithm as follows.

Definition 1 *The* greedy algorithm *chooses the query to ask at node v from the set* $\arg\max_{q \in Q} \sum_{a \in A} P(a|I_v, q) \sum_{o \in O} \min_{\theta \in U(I_v, q, a)} p(\theta, o)$.

The greedy algorithm does what we intend:

Theorem 2 *The greedy algorithm chooses a query that maximizes the probability of exiting immediately after it.*

Proof: By Corollary 1, we know that if we are currently at node v and do not exit early, and we transition to node w, then the probability of exiting early at node w is $1 - \frac{1 - \sum_{o \in O} \min_{\theta \in I_w} p(\theta, o)}{1 - \sum_{o \in O} \min_{\theta \in I_v} p(\theta, o)}$. Thus, if we ask query $q \in Q$ at node v, the probability of exiting immediately after q is $\sum_{a \in A} P(a|I_v, q)(1 - \frac{1 - \sum_{o \in O} \min_{\theta \in U(I_v, q, a)} p(\theta, o)}{1 - \sum_{o \in O} \min_{\theta \in I_v} p(\theta, o)}) = 1 - \frac{1}{1 - \sum_{o \in O} \min_{\theta \in I_v} p(\theta, o)}(1 - \sum_{a \in A} P(a|I_v, q) \sum_{o \in O} \min_{\theta \in U(I_v, q, a)} p(\theta, o))$. Choosing $q \in Q$ to maximize this expression is equivalent to choosing $q \in Q$ to maximize $\sum_{a \in A} P(a|I_v, q) \sum_{o \in O} \min_{\theta \in U(S_v, q, a)} p(\theta, o)$, which is exactly what the greedy algorithm does. ■

Theorem 3 *The greedy algorithm chooses the query for node v in time $O(|Q| \cdot |A| \cdot |O| \cdot |\Theta|)$.*

Unfortunately, the greedy algorithm can be arbitrarily far from optimal (even when all queries have equal cost):

Proposition 1 *There exist single-stage mechanisms S for which the greedy algorithm achieves only an arbitrarily small fraction of the possible query savings (even when S is deterministic, there are only three players, two types per player, and three outcomes; alternatively, even when priors over types are uniform, there are only three players, two types per player, and five outcomes).*

4.2.2. Dynamic programming algorithm Unlike the greedy algorithm, the dynamic program must build the entire tree. The program works by computing, for *every* possible information state I, the minimum possible expected number of queries $n(I)$ from that point on, *given* that we have not exited early. As before, let $U(I, q, a)$ be the information state that results from receiving answer a to q at I. Let $e(I, q, a)$ be the probability of exiting immediately after receiving answer a to q at I, given that we did not exit early at I. By Corollary 1, we can compute $e(I, q, a)$ as $1 - \frac{1 - \sum_{o \in O} \min_{\theta \in U(I, q, a)} p(\theta, o)}{1 - \sum_{o \in O} \min_{\theta \in I} p(\theta, o)}$. We obtain the recurrence

$$n(I) = min_{q \in Q} c(q) + \sum_{a \in A} P(a|I, q)(1 - e(I, q, a)) n(U(I, q, a))$$

Using the fact that $n(\{\theta\}) = 0$ for every type vector θ, we use this recurrence to compute the value of $n(I)$ for every I, starting with the small I and working up to larger ones.

Theorem 4 *The dynamic programming algorithm computes the value of $n(I)$ for all I in time $O(|Q| \cdot |A| \cdot |O| \cdot |\Theta| \cdot 2^{|\Theta|})$.*

We can retrieve the optimal multistage mechanism from this as follows: when we arrive at information state I and do not exit early, choose a query from $\arg\min_{q \in Q} \sum_{a \in A} P(a|I, q)(1 - e(I, q, a)) n(U(I, q, a))$.

5. Designing optimal multistage mechanisms

So far we have discussed how a given single-stage mechanism can be converted into an equivalent multistage mechanism. Here we will no longer take the single-stage design as a constraint. We develop a method for designing the single-stage mechanism in such a way that we get large savings in queries when we transform it into a multistage mechanism using the techniques described earlier. We focus on the case where the elicitation tree (query order) is given. It turns out that, using Corollary 1, we can directly integrate the eventual query savings into the linear programming formulation for AMD described in Subsection 2.1.

We say that node v is *on the elicitation path* for type vector θ if θ would lead us to ask the query at v (given that we do not exit early). For every internal node v in the tree, we add a term to the AMD objective (which maximizes the designer's objective) that indicates the probability of saving the query corresponding to this node.[10] (We say that we *save* the query corresponding to v when v is on the elicitation path, but we exit early at or before v.) Thus, the term in the objective for v is $c(v)P(v)e(v)$ where $c(v)$ is the cost of the query at node v, $P(v)$ is the probability of v being on the elicitation path, and $e(v)$ is the probability that we will exit early at or before v, given that v is on the elicitation path. $P(v)$ is a constant, but $e(v)$ is a variable that depends on how we set the outcome probabilities for the leaves. Specifically, by Corollary 1, we know that $e(v) = \sum_{o \in O} \min_{\theta \in S_v} p(\theta, o)$. The $\min$ operator is not linear, so we cannot add this expression to the LP objective directly. We work around this by letting $e(v) = \sum_{o \in O} e(v, o)$, where $e(v, o)$ is the probability of exiting early at or before v with outcome o, given that v is on the elicitation path. Then, for every $o \in O$ and $\theta \in S_v$, we add the constraint $e(v, o) \leq p(\theta, o)$.

Because linear programs can be solved to optimality in polynomial time, and the formulation above is polynomial in the number of outcomes and the number of types per agent (but not in the number of agents), the following theorem follows immediately:

Theorem 5 *The extension of the single-stage AMD formulation described above computes the optimal multistage mechanism for the given elicitation tree, taking query costs into account, in time polynomial in the number of outcomes and the number of types per agent (but not in the number of agents).*

This also begets an (inefficient) algorithm for generating the optimal multistage mechanism when neither the single-stage mechanism nor the elicitation tree is given: apply the above algorithm to every possible elicitation tree.

[10] Suitable scaling to ensure commensurability with the designer's objective is straightfoward; however, this does assume query costs can be accounted for additively.

6. Example application: optimal auctions

In this section, we demonstrate how our techniques can be applied to a simple auction example. In this auction, a single item is for sale. There are two bidders, A and B; the prior over each bidder's valuation for the item is uniform over $\{0, 1, 2, 3\}$. Our objective is to maximize expected revenue.

We first used the standard (single-stage) automated mechanism design software to generate an optimal single-stage dominant strategies mechanism for this setting. This produced the following mechanism (which is effectively the Myerson auction [18]):

	B bids 0	B bids 1	B bids 2	B bids 3
A bids 0	item not sold	item not sold	B wins, pays 2	B wins, pays 2
A bids 1	item not sold	item not sold	B wins, pays 2	B wins, pays 2
A bids 2	A wins, pays 2	A wins, pays 2	A wins, pays 2	B wins, pays 3
A bids 3	A wins, pays 2	A wins, pays 2	A wins, pays 2	A wins, pays 3

We then focused our attention on turning this single-stage mechanism into a multistage mechanism. We allowed only queries of the form "Is your valuation for the item at least k?" We observe that no matter which query is asked first, there is no chance of exiting after the first query. Hence, the greedy algorithm from Subsection 4.2 is underdetermined (and presumably will not perform very well). Instead, we used the dynamic programming algorithm from Subsection 4.2 to obtain the optimal elicitation tree for this single-stage mechanism. This produced the multistage mechanism in Figure 5.

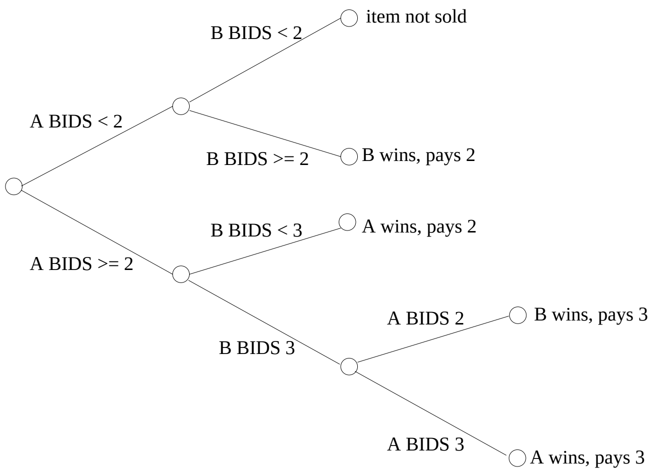

Figure 5. Single-stage auction converted to multistage.

We next studied whether any query gains could be made by changing the underlying single-stage mechanism while keeping the elicitation tree fixed, as described in Section 5.

We first placed a cost of 0.001 on each query. This produced the multistage mechanism in Figure 6.

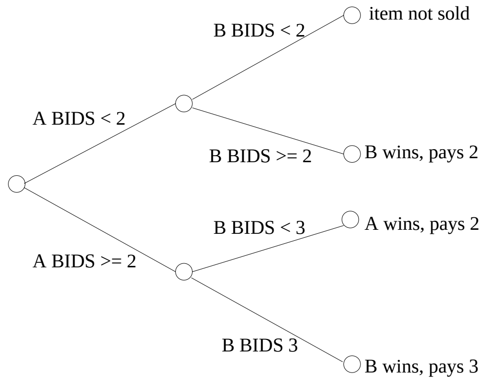

Figure 6. Underlying single-stage auction changed to save queries.

The only difference between this mechanism and the previous is how the tie is broken when both bidders bid 3. Therefore, this mechanism attains the same expected revenue as the previous mechanism. That is, the additional query savings obtained by this multistage mechanism come at no cost to the original objective.

Finally, we placed a cost of 0.5 on each query. This produced the multistage mechanism in Figure 7.

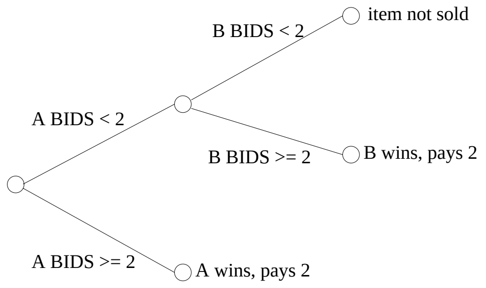

Figure 7. Underlying single-stage auction changed to save even more queries.

Effectively, this mechanism gives bidder A a take-it-or-leave-it offer of 2 for the item; if bidder A does not take this offer, the mechanism makes the same offer to bidder B. (Mechanisms that consist of sequences of take-it-or-leave-it offers have been studied systematically [22].) This mechanism in fact has lower expected revenue, but this loss in expected revenue is outweighed by the query savings that are obtained.

7. Mechanisms without dominant strategies

So far, we have restricted our study to multistage mechanisms whose single-stage correspondents have truth-telling as a dominant strategy. As discussed, this is helpful because in such multistage mechanisms, telling the truth is an ex-post equilibrium, so we need not worry that information revealed to agents by the mechanism will introduce strategic behavior. Nevertheless, we may also be interested in converting single-stage mechanisms that do not have dominant strategies, such as BNE mechanisms, to multistage mechanisms (e.g., because such mechanisms can achieve a higher objective value than dominant-strategy mechanisms).

Here we present initial results on converting BNE mechanisms into multistage mechanisms. These results are negative: they show that restricting ourselves to particular natural classes of multistage mechanisms may come at a loss of optimality. Thus, to design optimal multistage mechanisms, we need to search a broader space of mechanisms.

Proposition 2 *Even when the primary objective is social welfare and we use BNE as our solution concept, there exist settings in which immediately revealing the result of every query incurs a loss in objective value.*

The next result that we establish is that restricting ourselves to mechanisms that always chooses the next query deterministically can come at a loss.

Proposition 3 *There exist settings in which: 1) The primary objective is social welfare; 2) The optimal single-stage BNE incentive compatible mechanism is unique; 3) The unique optimal (in terms of query savings) elicitation tree to ask the queries for this mechanism is not (even BNE) incentive compatible; 4) There exists an elicitation tree for this mechanism that randomizes over the next query selected, is (BNE) incentive compatible, and has almost the same query savings as the optimal elicitation tree (and thus strictly greater query savings than any deterministic (BNE) incentive-compatible elicitation tree for this mechanism).*

Proof: Let there be two agents, 1 and 2; let agent 1 have type set $\Theta_1 = \{\theta_1^1, \theta_1^2, \theta_1^3\}$ and let agent 2 have type set $\Theta_2 = \{\theta_2^1, \theta_2^2\}$. Let the outcome set be $O = \{o_1, o_2, o_3, o_4\}$. Let the utility functions be as follows:

$u_1(\theta_1^1, o_1) = 0$, $u_1(\theta_1^2, o_1) = 0$, $u_1(\theta_1^3, o_1) = 0$, $u_2(\theta_2^1, o_1) = 0$, $u_2(\theta_2^2, o_1) = 0$;

$u_1(\theta_1^1, o_2) = 3$, $u_1(\theta_1^2, o_2) = 1 + \epsilon$, $u_1(\theta_1^3, o_2) = -1$, $u_2(\theta_2^1, o_2) = 1$, $u_2(\theta_2^2, o_2) = -4$;

$u_1(\theta_1^1, o_3) = -4$, $u_1(\theta_1^2, o_3) = 1$, $u_1(\theta_1^3, o_3) = 3$, $u_2(\theta_2^1, o_3) = 2$, $u_2(\theta_2^2, o_3) = 3$;

$u_1(\theta_1^1, o_4) = -5$, $u_1(\theta_1^2, o_4) = -5$, $u_1(\theta_1^3, o_4) = 4$, $u_2(\theta_2^1, o_4) = -5$, $u_2(\theta_2^2, o_4) = 4$.

The unique social-welfare maximizing outcome in each case is: $o(\theta_1^1, \theta_2^1) = o_2$, $o(\theta_1^1, \theta_2^2) = o_1$, $o(\theta_1^2, \theta_2^1) = o_3$, $o(\theta_1^2, \theta_2^2) = o_3$, $o(\theta_1^3, \theta_2^1) = o_3$, $o(\theta_1^3, \theta_2^2) = o_4$.

Let the probability distributions over types be as follows: $\gamma_1(\theta_1^1) = 0.4$, $\gamma_1(\theta_1^2) = 0.1$, and $\gamma_1(\theta_1^3) = 0.5$. $\gamma_2(\theta_2^1) = \gamma_2(\theta_2^2) = 0.5$. Choosing the social welfare maximizing outcome in every case is BNE incentive compatible, so this is the unique optimal single-stage BNE mechanism. (We omit the proof of this fact due to space constraint.)

Let the cost of a query be a secondary objective to the social welfare objective (or let the cost of a query be very small), so that we still want to implement a mechanism that always chooses the social welfare maximizing outcome. Let there be three queries: the single query to agent 2, and two queries to agent 1: *Is your type θ_1^1?* and *Is your type θ_1^3?* No single query is enough to determine the outcome for the optimal single-stage mechanism above. Disregarding strategic considerations, the unique optimal multistage mechanism corresponding to the optimal single-stage mechanism is:

```
Ask agent 2 for its type first;
  If it is $\theta_2^1$, ask agent 1 *Is your type $\theta_1^1$?*
    If so, choose $o_2$, otherwise choose $o_3$;
  If it is $\theta_2^2$, ask agent 1 *Is your type $\theta_1^3$?*
    If so, choose $o_4$, otherwise ask agent 1 *Is your type $\theta_1^1$?*
      If so, choose $o_1$, otherwise choose $o_3$.
```

This multistage mechanism needs all 3 queries only $0.5 \cdot 0.5 = 0.25$ of the time. Unfortunately, under this multistage mechanism, agent 1 has a (slight) incentive to lie when its true type is θ_1^2, and the first query to it is *Is your type θ_1^1?* In this case, agent 1 knows that agent 2 reported θ_2^1, so falsely answering *yes* would give it $1 + \epsilon$ rather than 1. Any other multistage mechanism that corresponds to the optimal single-stage mechanism and does not randomize over queries will cost us significantly in queries. But, instead, we can also slightly modify the given multistage mechanism and with very small probability ask agent 1 *Is your type θ_1^1?* after agent 2 answers θ_2^2. Now, when agent 1 is confronted with *Is your type θ_1^1?* as the first query while its true type is θ_1^2, there is a slight chance that agent 2 answered θ_2^2, in which case agent 1 does not want to answer *yes* and get o_1 rather than o_3. Because the benefit of lying in the other case is so small, it is outweighed, and agent 1 will answer truthfully. All other queries are still answered truthfully—thus this mechanism is (BNE) truthful. ■

A potential alternative to randomization by the mechanism is to obtain the randomization from mixed (i.e., randomized) strategies of the agents in mechanisms that are not truthful direct-revelation mechanisms.

8. Conclusions and future research

We extended the constrained-optimization based techniques for automated mechanism design to the design of multistage mechanisms, allowing reduction in elicitation burden by querying agents sequentially, and only querying them for information that is relevant given previous query responses. We focused primarily on the design of multistage mechanisms that correspond to dominant-strategy single-stage mechanisms, since these ensure truth-telling is an *ex-post* equilibrium (no matter what is revealed about other agents' answers). We described several techniques for converting single-stage mechanisms into multistage, both with and without fixed elicitation trees, and also showed how to augment single-stage AMD to produce single-stage mechanisms that can be maximally exploited in the conversion to multistage. We illustrated all of these techniques on an optimal auction example. Finally, we presented negative results on the design of multistage mechanisms that do not correspond to dominant-strategy single-stage mechanisms.

Future research includes developing techniques for designing multistage mechanisms that do not correspond to dominant-strategy single-stage mechanisms. We also hope to develop more efficient algorithms for optimal design when neither the single-stage mechanism nor the elicitation tree is given. Another direction is to extend techniques from machine learning to the query selection problem in our setting. Finally, we also plan to discover more efficient problem-specific techniques for the automated design of multistage mechanisms.

References

[1] 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.

[2] Gal Bahar and Moshe Tennenholtz. Sequential-simultaneous information elicitation in multi-agent systems. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Edinburgh, UK, 2005.

[3] Avrim Blum, Jeffrey Jackson, Tuomas Sandholm, and Martin Zinkevich. Preference elicitation and query learning. *Journal of Machine Learning Research*, 5:649–667, 2004.

[4] Andrew Blumberg and Abhi Shelat. Searching for stable mechanisms: Automated design for imperfect players. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 8–13, San Jose, CA, USA, 2004.

[5] Liad Blumrosen and Noam Nisan. Auctions with severely bounded communication. In *FOCS*, pages 406–415, 2002.

[6] Liad Blumrosen, Noam Nisan, and Ilya Segal. Multi-player and multi-round auctions with severely bounded communication. In *ESA*, pages 102–113, 2003.

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

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

[9] Vincent Conitzer and Tuomas Sandholm. Vote elicitation: Complexity and strategy-proofness. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 392–397, Edmonton, Canada, 2002.

[10] Vincent Conitzer and Tuomas Sandholm. An algorithm for automatically designing deterministic mechanisms without payments. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 128–135, New York, NY, USA, 2004.

[11] Vincent Conitzer and Tuomas Sandholm. Computational criticisms of the revelation principle. In *The Conference on Logic and the Foundations of Game and Decision Theory (LOFT-04)*, Leipzig, Germany, 2004.

[12] 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)*, pages 132–141, New York, NY, 2004.

[13] Claude d'Aspremont and Louis-Andre Gérard-Varet. Incentives and incomplete information. *Journal of Public Economics*, 11:25–45, 1979.

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

[15] E Kushilevitz and N Nisan. *Communication Complexity*. Cambridge University Press, 1997.

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

[17] Eric Maskin and John Riley. Optimal multi-unit auctions. In Frank Hahn, editor, *The Economics of Missing Markets, Information, and Games*, chapter 14, pages 312–335. Clarendon Press, Oxford, 1989.

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

[19] David Parkes. Iterative combinatorial auctions. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 3. MIT Press, 2006.

[20] Motty Perry and Philip J. Reny. A general solution to King Solomon's dilemma. *Games and Economic Behavior*, 26:279–285, 1999.

[21] Tuomas Sandholm and Craig Boutilier. Preference elicitation in combinatorial auctions. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 10. MIT Press, 2006.

[22] Tuomas Sandholm and Andrew Gilpin. Sequences of take-it-or-leave-it offers: Near-optimal auctions without full valuation revelation. In *Agent-Mediated Electronic Commerce (AMEC) workshop*, Melbourne, Australia, 2003.

[23] Rann Smorodinsky and Moshe Tennenholtz. Sequential information elicitation in multi-agent systems. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, Banff, Canada, 2004.

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