Using value queries in combinatorial auctions

Benoît Hudson* and Tuomas Sandholm

Carnegie Mellon University
Computer Science Department
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
{bhudson,sandholm}@cs.cmu.edu

Abstract. Combinatorial auctions, where bidders can bid on bundles of items, are known to be desirable auction mechanisms for selling items that are complementary and/or substitutable. However, there are $2^k - 1$ bundles, and each agent may need to bid on all of them to fully express its preferences. We address this by showing how the auctioneer can recommend to the agents incrementally which bundles to bid on so that they need to only place a small fraction of all possible bids. These algorithms impose a great computational burden on the auctioneer; we show how to speed them up dramatically. We also present an optimal elicitor, which is intractable but may be the basis for future algorithms. Finally, we introduce the notion of a *universal revelation reducer*, demonstrate a randomized one, and prove that no deterministic one exists.

1 Introduction

Combinatorial auctions, where agents can submit bids on *bundles* of items, are economically efficient mechanisms for selling k items to n bidders, and are attractive when the bidders' valuations on bundles exhibit *complementarity* (a bundle of items is worth more than the sum of its parts) and/or *substitutability* (a bundle is worth less than the sum of its parts). Determining the winners in such auctions is a complex optimization problem that has recently received considerable attention (e.g., [1, 5, 9, 13]).

An equally important problem, which has received much less attention, is that of bidding. There are $2^k - 1$ bundles, and each agent may need to bid on all of them to fully express its preferences. This can be undesirable for any of several reasons: determining one's valuation for any given bundle can be computationally intractable [12]; there is a huge number of bundles to evaluate; communicating the bids can incur prohibitive overhead (e.g., network traffic); and agents may prefer not to reveal all of their valuation information due to reasons of privacy or long-term competitiveness. Appropriate bidding languages [5,6,9,13,14] can solve

* The first author of this paper is a student. The material in this paper is based upon work supported by the National Science Foundation under CAREER Award IRI-9703122, Grant IIS-9800994, ITR IIS-0081246, and ITR IIS-0121678. Some of the results in this paper appeared in the AMEC 2002 workshop.

the communication overhead in some cases (when the bidder's utility function is compressible). We focus on the case where agents' valuation functions are largely incompressible. In these cases, when the number of items for sale is even moderate, the bidders will not be able to fully express their preferences. Instead, they may wastefully bid on bundles they will not win, and they may miss a better bid that they would have won.

Selective incremental preference elicitation by the auctioneer was recently proposed to address these problems [3]. Some of the theoretical properties of the so-called rank-lattice based elicitors were recently studied [4]. That family of elicitors suffers from having to ask queries in an extremely rigid order. In this paper we study the more practical approach where the elicitor uses value queries in arbitrary order. We first develop a framework to describe our algorithms (Section 2). Using this framework, we design a series of increasingly effective elicitation algorithms and study their properties theoretically and experimentally, showing that elicitation reduces revelation drastically, and that this benefit increases with problem size (Section 3). In Section 4 we provide an omniscient elicitor, which acts as an instance-specific lower bound on any real elicitor. We then discuss an optimal non-omniscient elicitor (Section 5) which is unfortunately not tractable. Finally, we introduce the notion of a universal revelation reducer, and demonstrate that while a randomized one exists, no deterministic one does (Section 6).

2 Elicitation framework

We study a setting where one auctioneer is selling a set K, $|K| = k$, of items to n bidders. We work with the standard independent private values model, that is, we assume that each bidder's valuations for the bundles are independent of the other bidders' valuations.

The elicitation framework that we use is a slightly modified version of that of [3]. The auctioneer asks questions of the agents (it *elicits* information, and thus it is an *elicitor*), and assimilates the information from the replies. Based on that information, the elicitor clears the auction if possible; if not, it elicits further information.

2.1 Value propagation

In this paper we focus on elicitation using *value queries*. In a value query, the elicitor asks an agent i for that agent's valuation $v_i(b)$ for bundle b.[1] Throughout

[1] An important issue is that the agents might reveal their valuations insincerely. Furthermore, the elicitor's queries leak information: a query that an agent receives signals to that agent information about how the other agents have answered the queries so far. That opens the door for additional strategic manipulation by the agents. However, if *Vickrey-Clarke-Groves (VCG)* pricing [2] is used (this requires some additional elicitation, but not significantly more), and the bidders' utility functions are quasilinear, then revealing one's preferences truthfully is an *ex post* equilibrium [3].

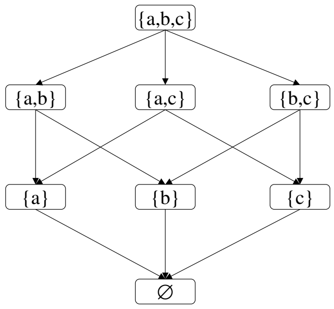

Fig. 1. Constraint network with 3 items a, b, and c.

this paper we make the usual assumption of *free disposal*, that is, any agent can throw away extra items for free, so extra items never decrease an agent's valuation. This allows the elicitor to infer an upper bound on any sub-bundle b^- of b: namely, no smaller bundle can be worth more than a larger bundle. Therefore $UB_i(b^-) = \min_{b \supseteq b^-}[v_i(b)]$. Similarly, no larger bundle can be worth less than a smaller bundle. Therefore, $LB_i(b^+) = \max_{b \subseteq b^+}[v_i(b)]$. The empty bundle is assumed to be worth 0.

To support fast inference of the upper and lower bounds, we implemented an interval constraint network. The elicitor maintains one network for each agent. In each network i, a node corresponds to a bundle b. The node is labeled by an upper bound $UB_i(b)$ and a lower bound $LB_i(b)$ which together define an interval in which the true value $v_i(b)$ is known to lie. A node for b has a directed link to a node for b' if $b \supseteq b'$. When a value for b becomes known, the elicitor sets $UB_i(b) = LB_i(b)$ to that value. It then propagates the value down the links to smaller bundles, enforcing the invariant that the $UB_i(b^-)$ in the network corresponds to the definition of the previous paragraph; similarly, it propagates the value up the links to larger bundles to enforce $LB_i(b^+)$.

2.2 Certificates

The auctioneer clears the auction if, given the information it has received, the auctioneer can infer that one allocation is worth at least as much as any other. That allocation is an optimal allocation. If the information the auctioneer has allows this inference, the information forms a *certificate* for that allocation. The certificate contains a set of value queries. A *minimal certificate* is a certificate that would cease to be a certificate if any query were removed from it. A *shortest certificate* is a certificate that has the smallest number of queries among all certificates. This depends on the problem instance. For any given instance, we call the length of the shortest certificate q_{min}.

2.3 Elicitation algorithm

The elicitation algorithm maintains a set of feasible allocations of items to bidders. The elicitor repeatedly chooses a query to make based on the information received so far, makes the query, and updates the set, pruning out provably suboptimal candidate allocations. The algorithm ends when there is only one allocation left.

SOLVE()
$C \leftarrow$ INITIALCANDIDATES(n, k)
while not DONE(C)
    do $q \leftarrow$ SELECTQUERY(C)
        ASKQUERY(q)
        $C \leftarrow$ PRUNE(C)

Here, C is a set of candidate allocations, where a *candidate* is a vector $c = \langle c_1, c_2, \ldots, c_n \rangle$ of bundles where the bundles contain no items in common (since no item may be simultaneously allocated to two bidders). There are n agents bidding on k items. The value of a candidate is $v(c) = \sum_i v_i(c_i)$; $UB(c) = \sum_i UB_i(c_i)$ is an upper bound, and $LB(c) = \sum_i LB_i(c_i)$ a lower bound.

INITIALCANDIDATES generates the set of all candidates, which is the set of all n^k allocations of the k items to the n agents (some agents might not get any items).

PRUNE removes, one candidate at a time, each candidate that is dominated by a remaining candidate. A candidate c *dominates* another candidate c'—which we write as $c \succeq c'$—if the elicitor can prove, using what it currently knows, that the value of c must be at least as high as the value of c'.[2]

DONE returns true if each candidate in the set of remaining candidates C is provably optimal. This is the case either if C has only one element, or if all candidates in C have known value (that is, $\forall c \in C$, $UB(c) = LB(c)$). Because the algorithm has just pruned, it knows that if all candidates have known value, then they have equal value.

SELECTQUERY selects the next query to be asked. This function can be instantiated in different ways to implement different elicitation policies, as we will show.

ASKQUERY takes a query (b, i), asks agent i for its value of bundle b, then appropriately updates the constraint network. In particular, every superbundle b^+ has its lower bound be the tighter of $v_i(b)$ and the previous lower bound $LB_i(b^+)$; every subbundle b^- has its upper bound be the tighter of $v_i(b)$ and the previous upper bound $UB_i(b^-)$.

---

[2] This is the case if $\sum_{i=1}^{n} \delta_i(c_i, c_i') \geq 0$, where $\delta_i(b, b')$ is the greatest difference between $v_i(b)$ and $v_i(b')$ that can be proven given the information in the constraint network. Namely, if $b \supset b'$, then $\delta_i(b, b') = \max(0, LB_i(b) - UB_i(b'))$; otherwise, $\delta_i(b, b') = LB_i(b) - UB_i(b')$.

3 Elicitation policies

We designed and tested several elicitation policies. All of them compute a set of potential queries, and then randomly pick one. We present the policies starting from the simplest. We then incrementally add restrictions on the query set to constrain the random choice towards hopefully better queries.

Each experimental plot shows what fraction of all value queries (of which there are $n(2^k - 1)$) are made before the auctioneer finds an optimal allocation and can prove that it is optimal (that no other allocation is better). We call this fraction the *elicitation ratio*. In each plot, each point represents an average over 50 runs, where each run is on a different problem instance (different draw of valuations for the agents). Each algorithm was tested on the same set of problem instances. The procedure used to generate the test instances is described in the appendix.

3.1 Random elicitation policy

The first policy we investigate simply asks random value queries. In the beginning, we generate the set of all $n(2^k - 1)$ value queries. Whenever it is time to ask a query, the policy chooses a random query from the set, ignoring those it has already asked or for which the value can already be inferred.

We can show that if it is possible to save elicitation, then on average this policy saves elicitation:

Proposition 1. *Let $Q = n(2^k - 1)$ be the total number of queries, and let q_{min} be the number of queries in the shortest certificate. For any given problem instance, the expected number of queries that the random elicitation policy asks is at most $\frac{q_{min}}{q_{min}+1}(Q+1)$.*

Proof. Assume pessimistically that a query is either required to prove the optimal allocation or useless. Under this assumption, the analysis reduces to the following problem. We have r red "necessary" balls and b blue "useless" balls in a bag. We then randomly draw one ball at a time without replacement. The question is how many balls we expect to draw before all red balls have been drawn. Let $e(r, b)$ be this number. The base case is $e(0, b) = 0$, because there are no red balls to draw. In the general case, we pick one ball from the bag. With probability $r/(r + b)$, it is red, so the bag now has $r - 1$ red balls and b blue balls. Similarly, with probability $b/(r + b)$, it is blue, so the bag now has r red balls and $b - 1$ blue balls. Therefore, $e(r, b) = 1 + \frac{r}{r+b}e(r - 1, b) + \frac{b}{r+b}e(r, b - 1)$. It is easy to verify that $e(r, b) = \frac{r}{r+1}(b + r + 1)$ solves this recurrence. In the elicitation setting, $r = q_{min}$ and $r + b = Q$. The result follows. □

The upper bound given in the above proposition only guarantees relatively minor savings in elicitation (especially because q_{min} increases when the number of agents and items increases). However, we base the proof on the pessimistic assumption that there is only one minimal certificate (namely, the shortest certificate), and that all other certificates are extensions of the shortest one. When

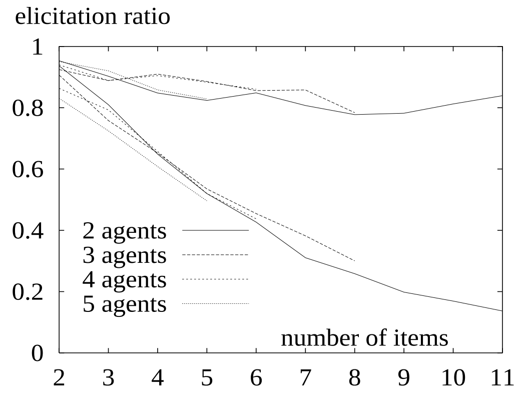

Fig. 2. *Top lines: Random elicitation policy. Bottom lines: Random allocatable policy. Each plot line corresponds to the elicitation ratio of a given number of agents as the number of items increases. Qualitatively, all lines for the same algorithm follow the same trend.*

there are several minimal certificates, it is more likely that the next query will complete a certificate. We do not know how to analyse how many minimal certificates we expect to be present. Therefore, we ran an experiment to see how well this policy does in practice. Figure 2 shows that the elicitation ratio q/Q is indeed less than 1. Also, the ratio slowly falls as the number of items increases, proving that indeed, the number of minimal certificates increases with the number of items. Nevertheless, we would hope to do better: on all the instance sizes we ran on, this algorithm asks more than half of all the queries.

3.2 Random allocatable elicitation policy

Essentially, the random elicitation policy is asking many queries which, as it turns out, are not useful. We will now present a useful restriction on the set of queries from which the elicitation policy should choose. The key observation is that the elicitor might already know that a bundle b will not be allocated to a particular bidder i – even before the elicitor knows precisely the bidder's valuation for the bundle. This occurs when the elicitor knows of a different allocation which it can prove will generate at least as much value as any allocation that allocates b to agent i. On the other hand, if the elicitor cannot (yet) determine this, then the bundle-agent pair is deemed *allocatable*.

Definition 1. *A bundle-agent pair* (b, i) *is* allocatable *if there is a remaining candidate allocation* $c \in C$ *such that* $c_i = b$*.*

We ran experiments (Figure 2) to determine whether the restriction helps in practice, and found that it clearly does. For example, at $k = 8$ items and $n = 3$ agents, the elicitation ratio of the unrestricted random policy is 78% while that of the allocatable-only policy is merely 30%. That is, the random elicitation policy restricted to eliciting only allocatable (b, i) avoids much of the elicitation needed in the unrestricted random elicitation policy.

3.3 Stronger restrictions

We analyzed several policies which restricted the query set more strongly than simply to allocatable bundles, trying to cleverly account for many features. A list of negative results includes: counting the number of allocations that include (b, i)[3]; counting the expected amount by which bounds will change in the constraint networks when we elicit (b, i); counting the expected number of bounds that will change; counting the expected number of candidates that will be pruned after eliciting (b, i). For the policies that count the expected value of a quantity, we also tried the minimum and maximum value. All of these policies fared worse in terms of elicitation ratio than the random allocatable elicitation policy.

The heuristic of using the amount by which we expect bounds to change fared poorly when using exact value queries, but in the different setting of approximate value queries, it did better than random elicitation [7]. Furthermore, the next section will show that this idea, combined with another heuristic, yields the best algorithm we know. It is not clear to the authors why it does poorly when used in its barest form.

In experiments of the random elicitation policy, the number of live candidates follows a sigmoidal curve whereby at first, almost no candidates are pruned. Then, the number of candidates falls precipitously. Finally, it takes several more queries to prune the last few candidates. Therefore, we conjecture that the heuristic of picking the query that will prune the most candidates fails to help because at first, it behaves almost identically to random elicitation. In the middle, almost any query reduces the candidate set dramatically, so the heuristic does not much improve the queries asked. Finally, at the end, almost all allocatable queries must be asked regardless of the heuristic. In fact, at all stages the heuristic may be biasing towards queries that prune candidates now but do not help prune more candidates later.

3.4 High-value candidate elicitation policy

We did find one policy that significantly outperforms the random allocatable elicitation policy. This policy is a modification of the elicitation policy that was recently used in a combinatorial exchange for allocating tasks in a multi-robot system [15]. The intuition is that to prove an allocation optimal, we must prove a sufficient high lower bound on it, while at the same time proving sufficient low upper bounds on all other allocations. By only picking from high-value candidates, we expect to be biasing toward asking questions that will need to be asked anyway. In addition, by picking from those queries that will reduce as many values as possible, we bias toward reducing upper bounds, which is desirable since

[3] This is the value-query policy previously proposed in [3].

there is typically only one optimum out of the n^k total candidates (the latter restriction was not present in the previous work).

Specifically, let $C_{\max}$ be the set of candidates of greatest upper bound ("high-value candidates"). That is, $C_{\max} = \{c \in C \text{ s.t. } UB(c) = \max_{c' \in C} UB(c')\}$. For each $(b, i) \in C_{\max}$ define subbundles(b, i) to be the number of other bundles in high-value candidates whose value might be affected upon eliciting $v_i(b)$. The subbundles are those $(b', i) \in C_{\max}$ for which $b \supset b'$ and $LB_i(b) < UB_i(b')$. Finally, pick uniformly at random among the (b, i) with the most subbundles.

In terms of elicitation ratio, this is the best policy we have yet developped. It achieves an elicitation ratio of only 24% with $k = 8$ items and $n = 3$ agents, as opposed to 30% for the random allocatable policy and 78% for the unrestricted random policy.

Representing candidates implicitly The policy of the previous section works well in terms of elicitation, but in terms of time it scales poorly with the number of agents. The chief cost is due to representing the candidates explicitly: PRUNE runs in time quadratic in the number of candidates, while SELECTQUERY and DONE run in time linear in the number of candidates, of which there are as many as n^k. Since the policy chooses among a set of size at most $n2^k$, we might hope to save work by implicitly representing the candidates (as long as $n > 2$).

We accomplish this by repeatedly solving an integer program (IP) every time a query is to be selected—rather than explicitly representing the set of candidates. We use the following IP to compute the value of the highest-valued candidate:

$$\begin{array}{lll} \text{maximize} & \sum_{i \in N, b \in 2^K} UB_i(b) x_i(b) & \\ \text{subject to} & x_i(b) \in Z_2 & \forall i \in N, \forall b \in 2^K \\ & \sum_{b \in 2^K} x_i(b) \leq 1 & \forall i \in N \\ & \sum_{i \in N} \sum_{b \ni j} x_i(b) \leq 1 & \forall j \in K \end{array}$$

The first constraint of the IP states that each bundle is either allocated or not. The second constraint states that each agent only gets one bundle, and the third constraint states that each item is allocated to only one agent. This is an assignment problem and can thus be solved quite quickly.

Upon solving the IP, the elicitor will know the value $UB_{\max}$ of the candidates with greatest upper bound. Then, for each pair (b, i) in turn, we force $x_i(b) = 1$ and solve again. This returns the value $UB_{\max}(b, i)$ of the candidates with greatest upper bound, constrained to only those candidates which allocate b to agent i. If $UB_{\max}(b, i) = UB_{\max}$, then (b, i) is in a high-value candidate. The elicitor now has the set of (b, i) that are in high-value candidates, and can proceed as before (that is, for each such (b, i), count the subbundles(b, i) and pick a random (b, i) among the ones with the greatest number of subbundles).

Implemented naively, the policy solves the IP for each pair (b, i) in each call to SELECTQUERY. One can sometimes avoid solving the IP by caching cache$(b, i) = UB_{\max}(b, i)$. Since the IP uses the current upper bounds on the true valuations, and upper bounds only decrease, the value of the IP solutions will only decrease.

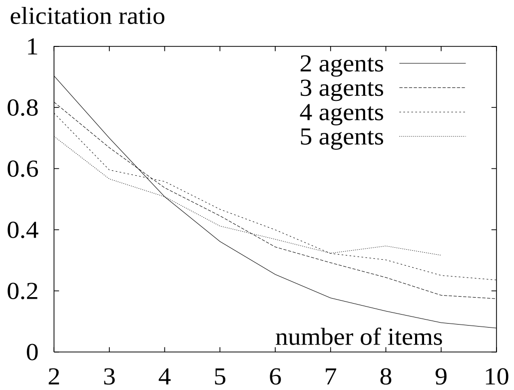

Fig. 3. *High-value candidate elicitation policy, with implicit candidate set representation. The legend is in the order of the plot lines at 2 items; the order reverses as the number of items grows. The elicitation ratio falls with increasing number of items, but grows with increasing number of agents, when there are more items than agents.*

That is, in a later call to SELECTQUERY, it will be the case that $UB_{\max}(b,i) \leq \text{cache}(b,i)$. Therefore, if $\text{cache}(b,i) < UB_{\max}$, the elicitor can infer that (b,i) is not in a high-value candidate, without computing $UB_{\max}(b,i)$.

Our experiments show that in our implementation, the implicit representation of candidates is faster than the explicit one already with three agents ($n = 3$). With 5 agents, the implicit approach is several orders of magnitude faster.

4 Omniscience

So far we have shown that the elicitors that we designed save most of the preference revelation. However, the question remains: what is the best that any elicitor could do?[4] To answer this question, we examine the setting where the elicitor is *omniscient*: it knows every agent's valuation function exactly. However, it must prove to a non-omniscient observer (say, the Federal Trade Commission) that it is conducting the auction correctly. That is, it must provide a certificate consisting of value queries and their answers. The elicitor tries to minimize the length of the certificate. This corresponds to the normal non-omniscient elicitor above which tries to ask as few queries as possible. Therefore, the performance of the omniscient elicitor provides a lower bound on that of any real—i.e., non-omniscient—elicitor.

We implemented a routine based on the IDA* search algorithm to find the shortest certificate. Firstly, we use the fact that there exists a shortest certificate

[4] It is known that *in the worst case*, exponential communication is required to find an (even approximately) optimal allocation in a combinatorial auctions—no matter what query types and elicitation policy is used [10].

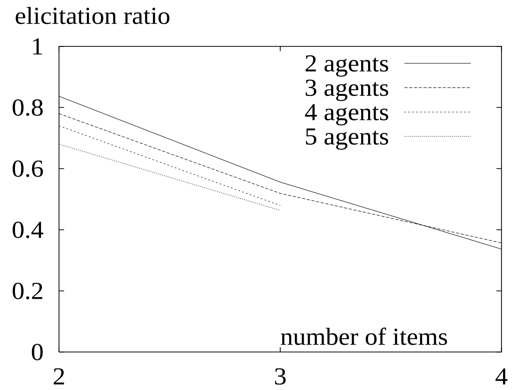

Fig. 4. *Omniscient elicitor. The results end with very few items because the search space size is doubly-exponential in the number of items.*

where the grand bundle value is queried from every agent [7]. Therefore, we immediately put (K, i) into the certificate for every agent i.

In the search, we define a lexicographic order on queries. This means that the search space is a tree, rather than a graph, eliminating any revisiting of nodes.

As is usual for IDA* search, we developed an admissible heuristic h for guiding the search. At any given node of the search, the h-value gives an lower bound on the number of further queries needed to obtain a certificate. We based our heuristic on the observation that for a certificate to prove that the optimal candidate dominates *all* candidates, it must at least prove that the optimal candidate dominates *any given* candidate. For every candidate c, we compute how many queries would be needed to separate c from the optimal candidate. The heuristic function h is the maximum over all c.[5]

This is clearly a weak heuristic: h is at most $2n$ and does not vary as the number of items grows, whereas we expect the certificate length to grow exponentially in the number of items! Instead, the major strength of the heuristic lies in its ability to discover that no certificate is reachable. Future work involves developing stronger heuristics.

If there are multiple optimal candidates, then we run IDA* for each optimum, in parallel, and use the best solution.

Figure 4 shows the elicitation ratio of the omniscient elicitor. Given these results, we can evaluate how much we might hope to improve the real (non-omniscient) elicitation algorithms presented above. The answer is that there remains room for improvement. For example, with 3 agents and 4 items, the high-value candidate elicitation policy launches 50% of all queries, and the omniscient elicitor launches 33% of all queries. However, it should be remembered that it is

[5] Because of the lexicographic order on queries, it may be that there is a candidate c which cannot be proven suboptimal using only queries still reachable in the current branch of the search space. In that case, the heuristic returns ∞ immediately.

highly unlikely that any non-omniscient policy could do as well as the omniscient algorithm.

5 Optimal elicitation algorithm

As we show in this section, one can also design an optimal non-omniscient elicitation algorithm. Unfortunately, it will be completely intractable even on the smallest instances. However, we hope it will motivate new, better elicitation algorithms.

Consider the game where the elicitor is asking questions, and the adversary answers them. The game ends when the revealed information constitutes a certificate. The goal of the elicitor is to ask as few questions as possible; the goal of the adversary is to force the elicitor to ask as many as possible. In order to guarantee the fewest number of queries, the elicitor can look through the game tree and pick the shortest path (from the root to a leaf). As we will see in the next section, the elicitor has no pure strategy against the adversary that does better than blindly eliciting the value of every bundle, but better mixed (i.e., probabilistic) strategies exist—for example, the unrestricted random policy.

In an auction, the elicitor is not playing against an adversary, but rather against a random distribution. In its game tree search, it should therefore assume a probability distribution over possible answers instead of the worst-case answer.

Unfortunately, a full game tree search is infeasible: at the root, the elicitor can chose among $n(2^k - 1)$ queries. At the next level, the adversary can choose among MAXBID possible answers. For most of the answers, the elicitor has $n(2^k - 1) - 1$ queries left to ask. And so on. Even with very aggressive pruning, the combinatorics are almost certainly too large to solve this problem optimally except on the smallest instances.

6 Universal revelation reducers

So far we have presented elicitation policies that, on average over instances, save a large amount of preference revelation. Now we ask the question: Do there exist *universal* elicitors, that is, elicitors that save revelation on all instances (excepting those where even the omniscient elicitor must reveal everything)?

Definition 2. *(universal revelation reducer)*
A universal revelation reducer is an elicitation policy with the following property: given a problem instance, it can guarantee (always in the deterministic case; in expectation over the random choices in the randomized case) saving some elicitation over full revelation—provided the shortest certificate is shorter than full revelation. More formally, if $q_{min} < Q$, the policy makes $q < Q$ queries.

Theorem 1. *The unrestricted random elicitation policy is a universal revelation reducer.*

Proof. From Proposition 1 we have a bound on the number of queries asked in expectation. This bound is less than Q provided $q_{min} < Q$, so in expectation, the policy asks fewer than all the queries. □

Interestingly, *deterministic* universal revelation reducers do not exist:

Theorem 2. *No deterministic value query policy is a universal revelation reducer.*

We prove that no deterministic policy can guarantee saving any elicitation by constructing a fooling set. A minimal fooling set can be built for the case where there are 2 items and 2 agents; we show a somewhat more general result, that an equivalent fooling set can also be built for the case where there are 2 items, but $n \geq 2$ agents.

The fooling set R_{fool} consists of valuation functions of the form:

$$\begin{gathered} \forall i \in N \quad v_i(ab) = 2 \\ \forall a \in K, \forall i \in N \; v_i(a) \in \{0, 1\} \end{gathered}$$

And either (a):

$$\begin{aligned} \exists a \in K \text{ s.t. } & \textstyle\sum_i v_i(a) = 2 \textbf{ and} \\ & \textstyle\sum_i v_i(b) = 0 \end{aligned}$$

Or (b):

$$\begin{aligned} \exists i \in N \text{ s.t. } & v_i(a) = v_i(b) = 1 \textbf{ and} \\ & \forall j \neq i, v_j(a) = v_j(b) = 0 \end{aligned}$$

That is, either one of the items has value 1 to two agents, and no agent wants the other item; or one of the agents is happy with either item and no other agent wants only a single item. Thus, there are exactly two 1 values.

The socially optimal allocation is to give the entire set of items K to one of the agents. Otherwise, we can only get value 1 from the allocation.

Lemma 1. *Each instance in R_{fool} has a certificate that does not fully reveal the agents' valuations.*

Proof. (of Lemma 1)
Any instance in the fooling set can be solved by revealing all the zero values and no 1 values. If we have an instance of type (a), we have now revealed that any allocation of item a to an agent i, and item b to another agent j, has value at most $UB_i(a) + UB_j(b) = 2 + 0 \leq v_i(ab)$. Similarly for instances of type (b). Thus, we have a certificate. We necessarily have two bundles of value 1, and after revealing the zeros, an observer knows only that those two bundles have value at most two. Therefore, we have not revealed the values of all bundles.

Proof. (of Theorem 2)
The proof that no deterministic universal reducer exists operates under the

model that the adversary can choose the instance during the execution of the policy. Because the policy is deterministic, this is equivalent to having the adversary examine the policy and choose an instance *a priori*.

To the first query $v_i(S)$ where S is either a or b, the adversary will return 1.

From then on, the adversary will return 0, until the policy asks either $v_i(\bar{S})$ or until it asks $v_j(S)$ having already asked all $v_l(S)$ for $i \neq l \neq j$, at which point the adversary will return 1. In other words, the adversary forces the policy to ask both 1 values.

If the certificate thus chosen reveals both $v_i(S) = 1$ and $v_i(\bar{S}) = 1$, then it must reveal $v_j(S) = v_j(\bar{S}) = 0$ for all $j \neq i$. Otherwise, the allocation of S (resp. $\bar{S}$) to agent i and $\bar{S}$ (resp. S) to agent j has value up to $UB_i(S) + UB_j(\bar{S}) = 1 + 2 > 2$ which contradicts that we have a certificate. Thus the certificate must reveal the value of all bundles.

If the certificate instead reveals both $v_i(S) = 1$ and $v_j(S) = 1$, then it must reveal $v_l(\bar{S}) = 0$ for all l. Otherwise, we do not have a certificate. In addition, if the adversary chose to answer $v_j(S) = 1$, then the policy asked $v_l(S)$ for all $i \neq l \neq j$. Thus, the certificate reveals the value of all bundles. □

The theorem, as stated, proves that no deterministic value query policy saves on all instance sizes: given an algorithm, the adversary can provide an instance that fools it. The proof is slightly stronger: even if we require the adversary to provide an instance with n agents, the adversary can fool the elicitor. We conjecture the yet-stronger result that given any deterministic value query policy, any $n \geq 2$, and any $k \geq 2$, the adversary is still able to provide an instance of the appropriate size that fools the elicitor.

7 Conclusions and future research

The work presented here showed how to drastically reduce the amount of valuation information bidders in a combinatorial auction need to reveal (and thus the amount they have to compute). Our elicitors do relatively well even compared to an optimal omniscient elicitor. Future research involves developing yet more effective elicitation policies, for instance finding a good approximation to the optimal elicitation policy we presented. In addition, to scale to larger auctions, we plan to develop elicitor algorithms that run in less time and space. This may entail compromising the elicitation ratio.

An additional desirable feature in elicitation is fairness: we would like to distribute the work load evenly among the bidders. Experimentally, this is roughly satisfied by using randomized algorithms: all equally important queries have equal probability of being made. Thus, if all bidders have equally important information, then there is no bias.

We focused here on value queries only. It is not unlikely that some other query types could be used to some advantage [3, 4, 7]. Motivated by the same concerns as our research on elicitors, considerable research has been devoted to ascending auctions (e.g. [11, 16] Those mechanisms can be viewed as a special case of the elicitation framework, where the queries are demand queries: If these were the prices, what items would you buy from the auction? Future research includes more deeply understanding the connection between preference elicitation and ascending auctions, and, if appropriate, designing auction mechanisms that combine features of both families.

Appendix: Instance generator

Unfortunately, real data for combinatorial auctions are not publicly available.[6] Therefore, as in all of the other academic work on combinatorial auctions so far, we used randomly generated data. We first considered using existing benchmark distributions. However, the existing problem generators output instances with sparse bids, that is, each agent bids on a relatively small number of bundles. This is the case for the CATS suite of economically-motivated random problem instances [8] as well as for the other prior benchmarks [1,5,13]. However, in many real settings (spectrum auctions for instance), every bidder has positive value on every bundle (at least due to renting and reselling possibilities). In addition, the instances generated by many of the earlier benchmarks do not honor the free disposal constraints.

To capture these considerations, we developed a new benchmark problem generator. In each problem instance we generate, each bidder has a nonzero valuation for almost every bundle, and all valuations honor free disposal. Specifically, the generator assigns, for each agent in turn, integer valuations using the following routine. We impose an arbitrary maximum bid value MAXBID $= 10^7$ in order to avoid integer arithmetic overflow issues, while at the same time allowing a wide range of values to be expressed. Valuations generated with this routine exhibit both complementarity and substitutability.

GenerateBids(K)
$G \leftarrow$ new constraint network
$S \leftarrow 2^K$ (the set of all bundles)
impose free disposal constraints on G
$UB(K) \leftarrow$ MAXBID
while $S \neq \emptyset$
    do pick b uniformly at random from S
        $S \leftarrow S - b$
        pick $v(b)$ uniformly at random from $[LB(b), UB(b)]$
        propagate $LB(b) = UB(b) = v(b)$ through G

References

1. A. Andersson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. In *Proc. ICMAS*, pp. 39–46, 2000.
2. E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[6] Furthermore, even if the data were available, they would only have some bids, not the full valuation functions of the agents (because not all agents bid on all bundles).

3. W. Conen and T. Sandholm. Preference elicitation in combinatorial auctions: Extended abstract. In *Proc. ACM-EC*, pp. 256–259, 2001. A more detailed description of the algorithmic aspects appeared in the IJCAI-2001 Workshop on Economic Agents, Models, and Mechanisms, pp. 71–80.
4. W. Conen and T. Sandholm. Partial-revelation VCG mechanism for combinatorial auctions. In *Proc. AAAI*, pp. 367–372, 2002.
5. Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proc. IJCAI*, pp. 548–553, 1999.
6. H. Hoos and C. Boutilier. Bidding languages for combinatorial auctions. In *Proc. IJCAI*, pp. 1211–1217, 2001.
7. B. Hudson and T. Sandholm. Effectiveness of preference elicitation in combinatorial auctions. In *AAMAS-02 workshop on Agent-Mediated Electronic Commerce (AMEC)*, 2002. Extended version: Carnegie Mellon University, Computer Science Department, CMU-CS-02-124, March 2002.
8. K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proc. ACM-EC*, pp. 66–76, 2000.
9. N. Nisan. Bidding and allocation in combinatorial auctions. In *Proc. ACM-EC*, pp. 1–12, 2000.
10. N. Nisan and I. Segal. The communication complexity of efficient allocation problems, 2002. Draft. Second version March 5th.
11. D. C. Parkes. iBundle: An efficient ascending price bundle auction. In *Proc. ACM-EC*, pp. 148–157, 1999.
12. T. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proc. AAAI*, pp. 256–262, 1993.
13. T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002. First appeared at the First International Conference on Information and Computation Economies, 1998. Also: Washington Univ., Dept. of Computer Science, tech report WUCS-99-01, January 28th, 1999. Also: *Proc. IJCAI*, pp. 542–547, 1999.
14. T. Sandholm. eMediator: A next generation electronic commerce server. *Computational Intelligence*, 18(4):656–676, 2002. Special issue on Agent Technology for Electronic Commerce. Early versions appeared in the Conference on Autonomous Agents (AGENTS-00), pp. 73–96, 2000; AAAI-99 Workshop on AI in Electronic Commerce, pp. 46–55, 1999; and as a Washington University, St. Louis, Dept. of Computer Science technical report WU-CS-99-02, Jan. 1999.
15. T. Smith, T. Sandholm, and R. Simmons. Constructing and clearing combinatorial exchanges using preference elicitation. In *AAAI-02 workshop on Preferences in AI and CP: Symbolic Approaches*, pp. 87–93, 2002.
16. P. R. Wurman and M. P. Wellman. AkBA: A progressive, anonymous-price combinatorial auction. In *Proc. ACM-EC*, pp. 21–29, 2000.