Effectiveness of Query Types and Policies for Preference Elicitation in Combinatorial Auctions [*]

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

Abstract

Combinatorial auctions, where agents can bid on bundles of items (resources, tasks, etc.), are desirable because the agents can express complementarity and substitutability among the items. However, expressing one's preferences can require bidding on all bundles. We evaluate an approach known as incremental preference elicitation [3] and show that as the number of items increases, the amount of information required to clear the auction is a vanishing fraction of the information collected in direct revelation mechanisms. Most of the elicitors also maintain the benefit as the number of agents increases. We prove that randomization helps, in that no deterministic elicitor is a universal revelation reducer. Finally, we present a new query type that allows agents to use anytime algorithms to give approximate answers that are refined only as needed.

1. Introduction

Combinatorial auctions (CAs), 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, but recent research has delivered winner determination algorithms that can optimally solve the problem for quite large numbers of items and bids in practice.

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: (1) there are a huge number of bundles to evaluate and communicate a bid; (2) determining one's valuation for any given bundle can be computationally intractable [9, 13, 16]; and (3) agents may prefer not to reveal all of their valuation information due to reasons of privacy or long-term competitiveness [15]. Appropriate bidding languages [6, 7, 11, 17, 18] can solve the communication overhead in some cases (when the bidder's valuation function is highly compressible). However, they still require the agents to completely determine and reveal their valuation functions and as such do not solve all the issues. So in practice, when the number of items for sale is even moderate, the bidders will not bid on all bundles. Instead, they may wastefully bid on bundles which they will not win, or fail to bid on bundles they would have won.

Incremental preference elicitation by the auctioneer was recently proposed to address these problems [3], but the idea was not evaluated. We implemented the most promising elicitation schemes from that paper, starting from a rigid search-based scheme, continuing to a general flexible elicitation framework. We then developed a host of new elicitation policies. We also provide theoretical results to guide the design of elicitation policies. Finally, we introduce and evaluate a new query type that takes the incremental nature of elicitation to a new level by allowing agents to give approximate answers that are refined only as needed.

It is known that even with free disposal and even with unlimited computation, finding an (even approximately) optimal allocation requires exponential communication [12] in the worst case. Nevertheless our experiments show that in practice elicitation reduces revelation drastically, and the benefit increases with problem size.

If elicitation is used in conjunction with *Vickrey-Clarke-Groves* pricing, each agent answering the elicitor's queries truthfully is an *ex post* equilibrium [3]. Determining the payments generally requires some additional elicitation, but experiments (omitted here due to limited space) show that the additional number of queries is only about 20%.

2. Auction and elicitation setting

We model the auction as having a single auctioneer selling a set K of items to n bidder agents (let $k = |K|$). Each

* This material 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. Prior versions of this paper appeared at AAMAS'02 workshop on Agent-Mediated Electronic Commerce and as Carnegie-Mellon University Technical Report CMU-CS-02-124.

agent i has a finite *valuation function* $v_i : 2^K \mapsto N$ that determines a private value $v_i(b)$ for each bundle $b \subseteq K$. We make the usual assumption that the agents have free disposal, that is, adding items to an agent's bundle never makes the agent worse off because, at worst, the agent can dispose of extra items for free. Formally, $\forall S \subseteq K, S' \subseteq S$, $v_i(S) \geq v_i(S')$. Many of the techniques of the paper can also be used without free disposal, although more elicitation is required due to less *a priori* structure.

At the start of the auction, the auctioneer knows the items and the agents, but has no information about the agents' value functions over the bundles—except that the agents have free disposal. The auction proceeds by having the auctioneer incrementally *elicit* value function information from the agents one query at a time until the auctioneer has enough information to determine an optimal allocation of items to agents. Therefore, we also call the auctioneer the *elicitor*. An allocation is optimal if it maximizes social welfare $\sum_{i=1}^{n} v_i(b_i)$, where b_i is the bundle that agent i receives in the allocation.The goal of the elicitor is to determine an optimal allocation with as little elicitation as possible, ideally without regard to computation. Clearly, the elicitor could conduct a straight-forward game tree search to decide what queries to ask. Against an adversary, we'll show that the adversary can force a bad outcome, but against nature (a random distribution), this search would be an optimal policy. Unfortunately, this is totally intractable. In order to be able to run experiments, we devise heuristic algorithms that run in time exponential in the number of items, and polynomial in the number of agents.

3. Elicitor's inference and constraint network

To minimize the number of queries, the elicitor must never ask a query whose answer could be inferred from the answers to previous queries. As per [3], to support the storing of information received from the agents, we have the elicitor store its information in a constraint network.Specifically, the elicitor stores a graph for each agent. In each graph, there is one node for each bundle b, labeled by an interval $[LB_i(b), UB_i(b)]$, which are respectively the tightest lower and upper bounds the elicitor can prove on the true $v_i(b)$ given the answers received to queries so far. We say a bound is *tight* when it is equal to the true value.

A directed edge (a, b) in the graph encodes the knowledge that the agent prefers bundle a over bundle b (that is, $v_i(a) \geq v_i(b)$). The elicitor may know this even without knowing $v_i(a)$ or $v_i(b)$: for example, the free disposal assumption creates implicit edges from every bundle a to every subbundle of a. An edge (a, b) lets the elicitor infer that $LB_i(a) \geq LB_i(b)$, which allows it to tighten the lower bound on a and on any of a's ancestors in the graph when the elicitor learns a new, tighter bound. Similarly, the elicitor can infer $UB_i(a) \geq UB_i(b)$, which allows it to tighten the upper bound on b and its descendants in the graph.

We define the relation $a \succeq b$ (read "a dominates b") to be true if we can prove that $v_i(a) \geq v_i(b)$. This is the case either if $LB_i(a) \geq UB_i(b)$, or if there is a directed path from a to b in the graph.

4. 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 queries and their answers. 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.

5. Rank lattice based elicitation

In this section we study the effectiveness of a technique proposed earlier [3,4]: *rank lattice based elicitation*, where the elicitor makes use of rank information about the bidders' bundles. Let $b_i(j)$, $1 \leq j \leq 2^k$, be the bundle that agent i has at rank j. In other words, $b_i(1)$ is the agent's most preferred bundle, $b_i(2)$ is its second most preferred, and so on down to $b_i(2^k)$, which is the empty bundle. The elicitor can put bounds on $v_i(b_i(j))$ using the constraint networks. Even without knowing $b_i(j)$ (which bundle it is that agent i values jth), it knows that $v_i(b_i(j-1)) \leq v_i(b_i(j)) \leq v_i(b_i(j+1))$. Thus an upper bound on $v_i(b_i(j-1))$ is an upper bound on $v_i(b_i(j))$, and a lower bound on $v_i(b_i(j+1))$ is a lower bound on $v_i(b_i(j))$.

The elicitor uses a *rank vector* $r = \langle r_1, r_2, \ldots, r_n \rangle$ to represent allocating $b_i(r_i)$ to each agent i. Not all rank vectors are feasible: the $b_i(r_i)$'s might overlap in items, which would correspond to giving the same item to multiple agents.

The set of all rank vectors defines a *rank lattice*. The *root* of the lattice is the all-ones rank vector; a child r' of a node r has all elements equal except one, which is incremented by one. A key observation in the lattice is that the children of a node have lower (or equal) value to the node. Given the rank lattice, we can employ search algorithms to find an optimal allocation. In particular, by starting from the root and searching in best-first order (always expanding the fringe node of highest value), we are guaranteed that the first feasible node that is reached is optimal. To know which rank vector in the fringe has highest value, the elicitor use the its constraint network to find bounds on the value of each rank vector. If it yet determine that one rank vector is worth more or at least as much as any other, it picks an arbitrary rank vector r and queries each agent i for $b_i(r_i)$ and $v_i(b_i(r_i))$.

6. Experimental setup

To evaluate the usefulness of elicitation, we conducted a host of experiments. We generated 50 instances of each size and ran the elicitation algorithms on those instances. Each point on the plots corresponds to the average performance over the 50 runs. The plots show results for those instance sizes on which the algorithms could solve every instance in under 2 minutes on a 2.8 GHz Intel machine.

Unfortunately, real data for CAs are not publicly available. Therefore, as in all of the other academic work on CAs so far, we used randomly generated data. 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 [10] as well as for many other prior benchmarks [1,6,17]. This is not necessarily realistic: while the bidders may far prefer some items and bundles to others, they will often have non-zero value on almost every bundle, at least due to reselling possibilities and, in some domains (such as spectrum or real estate auctions), renting. In addition, the instances generated by many of the earlier benchmarks do not exhibit free disposal.

The instances were generated by assigning, for each agent in turn, integer valuations using the routine below. We impose an arbitrary maximum bid value $\mathrm{MAXBID} = 10^7$ 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 and observe the free disposal assumption.

```
GENERATEBIDS($K$)
1  $G \leftarrow$ new constraint network
2  $S \leftarrow 2^K$ (the set of all bundles)
3  impose free disposal constraints on $G$
4  $UB(K) \leftarrow \mathrm{MAXBID}$
5  while $S \neq \emptyset$
6      pick $b$ uniformly at random from $S$
7      $S \leftarrow S - b$
8      pick $v(b)$ uniformly at random from $[LB(b), UB(b)]$
9      propagate $LB(b) = UB(b) = v(b)$ through $G$
```

7. Rank lattice experiments

We ran experiments to evaluate the efficiency of rank lattice based elicitation. Define the *elicitation ratio* to be the number of queries asked divided by the number of queries asked in full revelation. In full revelation, the number of queries is $n(2^k - 1)$ (that is, for each agent, one value query for each of the 2^k bundles except the empty bundle). Figure 1 shows that as the number of items in the auction increases, the elicitation ratio decreases quickly.

Unfortunately, Figure 1 also shows that as the number of agents n grows, the advantage from rank lattice based elicitation decreases. This can be explained as follows. As the number of agents increases, the average number of items that an agent wins decreases. Thus agents will usually win

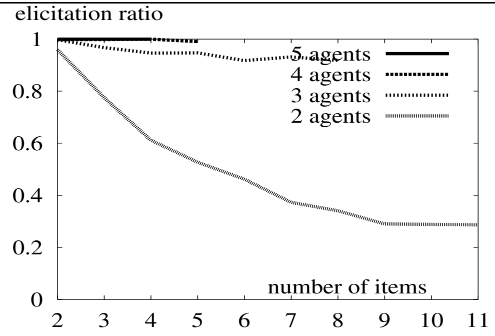

Figure 1. Performance of rank lattice based elicitation. The curves for 4 and 5 agents are barely visible, at elicitation ratio almost 1.

smaller, lower-ranked bundles. Because rank lattice based elicitors require the agents to reveal all high-rank bundles before any low-rank bundles, as the number of agents increases, each agent reveals a greater number of bundle values. This holds not only for the specific elicitor described above, but any elicitor that asks queries in order of rank (even if the elicitor had an oracle for deciding which queries should be asked from which agents). Furthermore, a recent result [4] implies that no rank lattice based elicitation algorithm is better on all instances than the elicitor tested here.

8. General elicitation framework

Due to these limitation of rank lattice based elicitors, we now move to a more general elicitation framework. As we show, this enables one to develop algorithms that ask significantly fewer queries and scale well in the number of agents.

The framework allows a richer set of query types (to accommodate for different settings where answering some types of queries is easier than answering other types); allows more flexible ordering of the queries at run time; and never considers infeasible solutions. The general elicitor template is a slightly modified version of that of Conen & Sandholm [3]:

```
SOLVE()
1  $C \leftarrow$ INITIALCANDIDATES$(n, k)$
2  while not DONE$(C)$
3      $q \leftarrow$ SELECTQUERY$(C)$
4      ASKQUERY$(q)$
5      $C \leftarrow$ PRUNE$(C)$
```

Here, C is a set of candidates 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. Unlike with rank vectors, all candidates are feasible. 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 $LB(c) = \sum_i LB_i(c_i)$ is 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 get no items).

PRUNE removes, one candidate at a time, each candidate that is dominated by a remaining candidate (a candidate c

dominates another candidate c' if the elicitor can prove that the value of c is at least as high as that of c').

DONE returns true if all remaining candidates in C are 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, asks the corresponding agent for the information, and appropriately updates the constraint network. The details of updating the network are discussed in conjunction with each query type below.

8.1. Value queries

The most basic query asks an agent i to reveal $v_i(c_i)$ exactly. We call such queries *value queries*. Upon receiving the answer, ASKQUERY sets $LB_i(c_i) = UB_i(c_i) = v_i(c_i)$ and propagates the new bounds upstream and downstream through the constraint network as described earlier.

8.1.1. Random elicitation policy A naive policy simply asks random value queries, ignoring those it has already asked or for which the value can already be inferred. If it is possible to save elicitation on the CA instance at hand, then on average this policy does:

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 $\frac{q_{min}}{q_{min}+1}(Q + 1)$.*

Most proofs are omitted due to limited space.

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 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: this elicitor still asks about 80% of all the queries.

8.1.2. 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*.

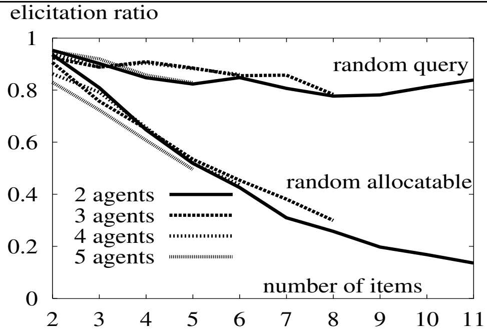

Figure 2. Top lines: Random query policy. Bottom lines: Random allocatable policy. Each plot line corresponds to the elicitation ratio of a given number of agents as the number of items increases.

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

Now we refine the random elicitation policy to ask queries on allocatable (b, i) only (and queries that have already been asked or whose answer can be inferred are still never asked). This restriction is intuitively appealing, but sometimes hurts:

Proposition 2 *There exist cases where, given the queries asked so far in the elicitation process, the minimal set of queries to complete the elicitation is smaller than the minimal set of queries to complete the elicitation when queries are restricted to allocatables.*

However, the restriction cannot hurt arbitrarily much:

Theorem 1 *No matter what queries have been asked so far, the smallest number of value queries to complete a certificate when restricted to allocatable queries is at most twice the number of value queries the random elicitor would need to complete a certificate.*

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%. Also, unlike in rank lattice based elicitation, the benefit from elicitation does not go away as the number of agents increases.

8.1.3. Stronger restrictions We analyzed several policies which restricted the query set more strongly than simply to allocatable queries, trying to cleverly account for many features. A list of negative results includes: the policy previously proposed in [3], namely counting the number of allocations that include (b, i); 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. 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 or no better than the random allocatable policy.

Under the random allocatable policy, if we track the number of candidates in C over time, we see three distinct phases: initially, almost no candidates are pruned regardless of what query was asked; then there is a brief phase where almost all candidates get pruned; and finally there is a phase where almost every query needs to be asked. This suggests (but does not prove) that any policy based on maximizing the number of pruned candidates should fail to significantly improve upon the random allocatable policy: the only time when any query could prune more than another is in the brief middle phase.

8.1.4. 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 [20]. 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 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 $\text{subbundles}(b, i)$ 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.

This is the best policy we have yet developed. 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 $\text{subbundles}(b, i)$ 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 $\text{cache}(b, i) = UB_{\max}(b, i)$. 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. 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.

8.1.5. The grand bundle should be queried Intuitively it is appealing to elicit from every agent the value for the grand bundle (i.e., the bundle that consists of all items) because that sets an upper bound on all bundle-agent pairs via

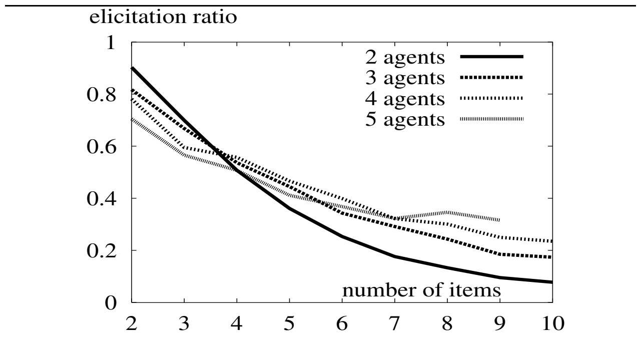

Figure 3. High-value candidate elicitation policy, with implicit candidate set representation. The legend is in the order of the plot lines at 2 items; the order reverses as the number of items grows. The elicitation ratio falls with increasing number of items, but grows with increasing number of agents,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

the free disposal assumption. We present here a proof that this is in fact almost always required. Namely, almost all instances require this upper bound from every agent, and those few instances that do not require it from every agent still require it from all but one agent. Therefore, the first thing all our elicitation policies query is the value of the grand bundle (or a bound on the value).

Proposition 3 *To determine an optimal allocation, any elicitation policy must prove an upper bound on $v_i(K)$ for every agent i to which the grand bundle K is* not *allocated.*

Furthermore:

Theorem 2 *Assume there are at least 2 bidders. There is a policy (possibly requiring an oracle for choosing the queries) using value queries that asks $v_i(K)$ for* every *agent i and that asks the fewest possible queries (among all elicitors that use value queries only).*

8.1.6. Omniscience and instance-specific lower bounds So far we have shown that the elicitors that we designed save most of the preference revelation compared to a direct revelation mechanism. However, it is perhaps more interesting to compare to the best possible performance on each instance. To bound this this, we examine an *omniscient* elicitor: 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. The number of queries asked by the omniscient elicitor provides an instance-specific lower bound on the number of queries asked by any real elicitor.

We implemented an IDA* search to find the shortest certificate. Unfortunately, the search space is double exponential in the number of items, so we could not scale this to more than 4 items. 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. This means there is some room for improvement. However, it is highly unlikely that any non-omniscient policy could do as well as the omniscient algorithm.

8.1.7. Universal revelation reducers So far we have presented elicitors 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 *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.*

Proposition 4 *The unrestricted random elicitation policy is a universal revelation reducer. (Immediate from Prop. 1).*

However, interestingly:

Theorem 3 *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:

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

And either (a):

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

Or (b):

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

That is, either (a) 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 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: 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. ■

Lemma 2 *Every deterministic algorithm has at least one instance in R_{fool} that makes it query every bundle and agent.*

Proof: The proof 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 $\mathit{UB}_i(S) + \mathit{UB}_j(\bar{S}) = 1 + 2 > 2$ 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. ■

Proof:[of Proposition 3] Lemma 1 shows that any instance of R_{fool} has $q_{min} < Q$. Meanwhile, Lemma 2 shows that any deterministic algorithm will make Q queries. Hence, no deterministic algorithm is a universal reducer. ■

8.2. Order queries

In some applications, agents might not know the values of bundles, and might need to expend a lot of effort to determine them [9, 16], but might easily be able to see that one bundle is preferable over another. In such settings, it would be sensible for the elicitor to ask *order queries*, that is, ask an agent i to order two given bundles c_i and c'_i (to say which of the two it prefers). The agent will answer $c_i \succeq c'_i$ or $c'_i \succeq c_i$ or both. ASKQUERY will then create new edges in the constraint network to represent these new dominates relations. By asking only order queries, the elicitor cannot compare the valuations of one agent against those of another, so it cannot determine a social welfare maximizing allocation. However, order queries can be helpful when interleaved with other types of queries.

8.3. Interleaving value and order queries

We developed an elicitor that uses both value and order queries. It mixes them in a straightforward way, alternating between the two (starting with an order query). When an order query is to be asked, the elicitor computes all tuples (a, b, i) where a and b are each allocated to agent i in some candidate, and where the elicitor knows neither $a \succeq b$ nor $b \succeq a$. The elicitor then picks randomly. When a value query is to be asked, it is chosen using the random allocatable policy.

The policy described above is able to reduce the number of precise values it elicits, by about 10%, over asking only value queries. Depending on the cost model, this may or may not be an improvement: it is an improvement if order queries cost less than 0.1 units (where value queries cost 1). This seems like a not unreasonable assumption: in many cases it should be far easier to compare two bundles than to find their exact value. More work needs to be done to choose better order queries, and to combine value and order queries.

Another advantage of the mixed value-order query policy is that it does not depend as critically on free disposal. Without free disposal, the policy that uses value queries only would have to elicit all values. The order queries in the mixed policy, on the other hand, can create useful edges in the constraint network which the elicitor can use to prune candidates.

8.4. Bound-approximation queries

In many settings, the bidders can roughly estimate valuations easily, but the more accurate the estimate, the more costly it is to determine. In this sense, the bidders determine their valuations using anytime algorithms [9]. For this reason, we introduce a new query type, a *bound-approximation query*: the elicitor asks an agent i to spend some time t to tighten the agent's upper bound $\mathit{UB}_i(b)$ (or lower bound $\mathit{LB}_i(b)$) on the value of a given bundle b. This query type leads to more incremental elicitation in that queries are not answered with exact information, and the information is refined incrementally on an as-needed basis.

Using randomly chosen bound-approximation queries as the elicitation policy would work, but the more sophisticated elicitation policy that we developed chooses the query that maximizes the benefit of receiving the information from that query. The benefit is defined to be the sum over all bundles in remaining candidates of the amount by which the bounds on each bundle will change given the new information. The elicitor optimistically hopes that the new bound z is such that it will change the most possible bounds: that is, when computing the benefit of a lower bound query, it assumes $z = \mathit{UB}_i(b)$ while when computing the benefit of an upper query, $z = \mathit{LB}_i(b)$ (the change in upper bound is always finite because we first elicit the grand bundle, as motivated Proposition 3). In this work, we did not intelligently choose a time bound: instead, we simply pass $t = 0.2$ for every query. Computing the expected benefit rather than the optimistic benefit gave very similar results. It is interest-

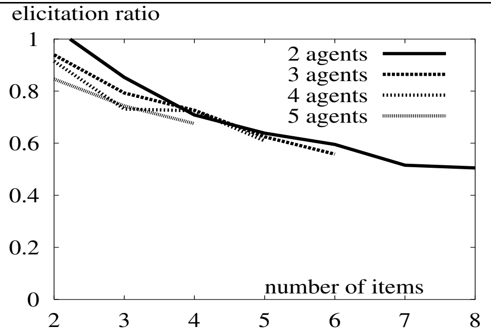

Figure 4. Elicitation using bound-approximation queries.

ing to note that, as we saw in Section 8.1.3, this policy did poorly at choosing value queries.

In the experiments, we needed a way of determining what answer an agent i might give to a query that asked it to spend t time refining a bound on b. We want to model diminishing returns to computation, as is usually the case with anytime algorithms. We therefore decided on the following (arbitrary) function: After spending a total of time t, $0 \leq t \leq 1$, the agent will report a lower bound of $v_i(b)\sqrt{t}$ or an upper bound of $(2 - \sqrt{t})v_i(b)$. Furthermore, we assume that the bound-tightening algorithm can be restarted without penalty. It is important to note that the elicitation algorithm does not know the details of this experiment: otherwise, it could spend ϵ time on a bound and immediately solve for $v_i(b)$, which is unrealistic.

Figure 4 shows that as the number of items increases, the fraction of the overall computation cost actually incurred diminishes: the optimal allocation is determined while querying only very approximate valuations on most bundle-agent pairs. The method also maintains its benefit as the number of agents increases.

9. Conclusions and future research

We presented the first experimental evaluation of preference elicitation in CAs. We developed a method for evaluation: against direct revelation, or against an omniscient elicitor. Through several theoretical and experimental advances, we developed the most effective elicitor to date, for general CAs. In all of the elicitors in this paper (except the unrestricted random one), as the number of items for sale increases, the amount of information elicited is a small fraction of the information needed for a traditional direct revelation mechanism. Each of the elicitation schemes (except the rank lattice based one) also maintains its benefit as the number of agents increases.

Current work includes studying new query types and elicitation policies, as well as applying elicitation to combinatorial reverse auctions and exchanges [20], both with and without free disposal. Future work also includes developing a better understanding of the relationship between preference elicitation and ascending CAs where the auction proceeds in rounds, and in each round the bidders react to price feedback by revealing demand (e.g., [14, 21]). This can be viewed as falling within our general preference elicitation framework, but with a different query class: *demand queries*. Very recently, interesting theoretical results on this connection have been proven [2, 8]: in some (but not other) restricted classes of CAs, a polynomial number of queries suffices when both demand and value queries are used. Such results have recently also been derived for value queries alone [5, 19, 22].

References

[1] A. Andersson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. *ICMAS*, 2000.

[2] A. Blum, J. Jackson, T. Sandholm, and M. Zinkevich. Preference elicitation and query learning. *COLT*, 2003.

[3] W. Conen and T. Sandholm. Preference elicitation in combinatorial auctions: Extended abstract. *ACM-EC*, 2001.

[4] W. Conen and T. Sandholm. Partial-revelation VCG mechanism for combinatorial auctions. *AAAI*, 2002.

[5] V. Conitzer, T. Sandholm, and P. Santi. Combinatorial auctions with k-wise dependent valuations. Draft, Oct. 2003.

[6] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. *IJCAI*, 1999.

[7] H. Hoos and C. Boutilier. Bidding languages for combinatorial auctions. *IJCAI*, 2001.

[8] S. Lahaie and D. Parkes. Applying learning algorithms to preference elicitation. *ACM-EC*, 2004.

[9] K. Larson and T. Sandholm. Costly valuation computation in auctions. *TARK VIII*, 2001.

[10] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. *ACM-EC*, 2000.

[11] N. Nisan. Bidding and allocation in combinatorial auctions. *ACM-EC*, 2000.

[12] N. Nisan and I. Segal. The communication requirements of efficient allocations and supporting Lindahl prices, 2003. Working Paper (version: March 2003).

[13] D. C. Parkes. Optimal auction design for agents with hard valuation problems. *Agent-Mediated Electronic Commerce Workshop at IJCAI*, 1999.

[14] D. C. Parkes and L. Ungar. Iterative combinatorial auctions: Theory and practice. *AAAI*, 2000.

[15] M. H. Rothkopf, T. J. Teisberg, and E. P. Kahn. Why are Vickrey auctions rare? *Journal of Political Economy*, 98(1):94–109, 1990.

[16] T. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. *AAAI*, 1993.

[17] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, Jan. 2002.

[18] T. Sandholm. eMediator: A next generation electronic commerce server. *Computational Intelligence*, 18(4):656–676, 2002. Special issue on Agent Technology for Electronic Commerce.

[19] P. Santi, V. Conitzer, and T. Sandholm. Towards a characterization of polynomial preference elicitation with value queries in combinatorial auctions. *COLT*, 2004.

[20] T. Smith, T. Sandholm, and R. Simmons. Constructing and clearing combinatorial exchanges using preference elicitation. *AAAI-02 workshop on Preferences in AI and CP: Symbolic Approaches*, 2002.

[21] P. R. Wurman and M. P. Wellman. AkBA: A progressive, anonymous-price combinatorial auction. *ACM-EC*, 2000.

[22] M. Zinkevich, A. Blum, and T. Sandholm. On polynomial-time preference elicitation with value queries. *ACM-EC*, 2003.