Generalizing Preference Elicitation in Combinatorial Auctions

Content Areas: multiagent systems, negotiation

Abstract

Combinatorial auctions where agents can bid on bundles of items are desirable because they allow the agents to express complementarity and substitutability between the items. However, expressing one's preferences can require bidding on all bundles. Selective incremental preference elicitation by the auctioneer was recently proposed to address this problem but the idea was not evaluated. In this paper we show that automated elicitation is extremely beneficial: as the number of items for sale increases, the amount of information elicited is a small and diminishing fraction of the information collected in traditional "direct revelation mechanisms" where bidders reveal all their valuation information. The elicitors also maintain the benefit as the number of agents increases—except rank lattice based elicitors which we show ineffective. We also develop elicitors that combine different query types, and we present 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 on an as-needed basis. We show that determining VCG payments requires very little additional elicitation. Finally, we show that elicitation can be easily adapted to combinatorial reverse auctions, where the benefits are similar to those in auctions, except that the elicitation ratio *improves* as the number of agents increases. In the process, we present methods for evaluating different types of elicitation policies.

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., [Fujishima *et al.*, 1999; Nisan, 2000; Sandholm, 2002a]).

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 [Parkes, 1999; Sandholm, 2000; Larson and Sandholm, 2001]; 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 [Rothkopf *et al.*, 1990]. Appropriate bidding languages [Fujishima *et al.*, 1999; Nisan, 2000; Hoos and Boutilier, 2001; Sandholm, 2002a; 2002b] can solve the communication overhead in some cases (when the bidder's utility function is compressible). However, they still require the agents to completely determine and transmit 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 cannot bid on all bundles. Instead, they may bid on bundles which they will not win, and they may fail to bid on bundles they would have won. The former problem leads to wasted effort, the latter to reduced economic efficiency of the resulting allocation of items to bidders.

A recent paper [Conen and Sandholm, 2001] proposed to have the auctioneer incrementally elicit the preferences but did not evaluate the idea. In this paper we build on those ideas, examining new elicitation policies and query types and evaluating them with mostly experimental and also some theoretical results. We also study the complexity not only of determining the optimal allocation of items to bidders but also determing the bidders' Vickrey-Clarke-Groves (VCG) payments [Groves, 1973]. In addition to combinatorial auctions we study combinatorial reverse auctions (procurement auctions). Our experiments show that only a very small fraction of the bidders' preference information needs to be elicited in order to determine the provably optimal allocation and the VCG payments.

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 agent i has a *valuation function* $v_i : 2^K \mapsto \mathbb{R}^+$ that determines a positive, finite, and 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, $\forall b \subseteq K, b' \subseteq b$, $v_i(b) \geq v_i(b')$. The techniques of the paper could also be used without free disposal, although more elicitation would be 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.[1]

[1]Social welfare can only be maximized meaningfully if bidders' valuations can be compared to each other. We make the usual assumption that the valuations are measured in money (dollars) and thus can be directly compared.

The goal of the elicitor is to determine an optimal allocation with as little elicitation as possible.[2]

3 Elicitor's constraint network

The elicitor, as we designed it, never asks a query whose answer could be inferred from the answers to previous queries. To support the storing and propagation of information received from the agents, we have the elicitor store its information in a constraint network.[3] Specifically, the elicitor stores a graph for each agent. In each graph, there is one node for each bundle b. Each node is labeled by an interval $[LB_i(b), UB_i(b)]$. The lower bound $LB_i(b)$ is the highest lower bound the elicitor can prove on the true $v_i(b)$ given the answers received to queries so far. Analogously, $UB_i(b)$ is the lowest upper bound. We say a bound is *tight* when it is equal to the true value.

Each graph can also have directed edges. A directed edge (a, b) 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 $v_i(b)$. 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. 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. The free disposal assumption allows the elicitor to infer the following dominance relations before the elicitation begins: $\forall b \subseteq K, b' \subseteq b, b \succeq b'$.

4 Rank lattice based elicitation

In this section we study the effectiveness of a technique proposed earlier [Conen and Sandholm, 2001; 2002]: *rank lattice based elicitation*. The idea is that the elicitor can make use of rank information about the bidders' bundles. Let $b_i(r_i)$, $1 \leq r_i \leq 2^k$, be the bundle that agent i has at rank r_i. In other words, $b_i(1)$ is the agent's most preferred bundle, $b_i(2)$ is its second most preferred bundle, and so on down to $b_i(2^k)$, which is the empty bundle.

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 value of a rank vector r is $v(b(r)) = \sum_i v_i(b_i(r_i))$.

The elicitor can put bounds on $v_i(b_i(r_i))$ using the constraint networks. Even without knowing $b_i(r_i)$ (which bundle it is that agent i values r_ith), it knows that $v_i(b_i(r_i - 1)) \leq v_i(b_i(r_i)) \leq v_i(b_i(r_i + 1))$. Thus an upper bound on $v_i(b_i(r_i - 1))$ is an upper bound on $v_i(b_i(r_i))$, and a lower bound on $v_i(b_i(r_i + 1))$ is a lower bound on $v_i(b_i(r_i))$.

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.

A complication arises in that the elicitor may have insufficient information to determine which node has highest value, and may need to effect further preference elicitation from the bidders. We implemented the following algorithm for doing this. It corresponds to an elicitation policy where as long as we cannot prove which node on the fringe is the best, we pick an arbitrary node and elicit just enough information to determine its value.

```
FINDBESTNODE(FRINGE)
1  $S \leftarrow$ FRINGE
2  remove from $S$ all $r$ dominated by some $r'$ in $S$
3  if all $r \in S$ have the same value
4     return arbitrary $r \in S$
5  choose $r \in S$ whose value we don't know exactly
6  for each agent $i$
7     if elicitor does not know $b_i(r_i)$
8        ask agent $i$ what bundle it ranks $r_i$th
9     if elicitor does not know $v_i(b_i(r_i))$ exactly
10       ask agent $i$ for its valuation on bundle $b_i(r_i)$
11 goto 2
```

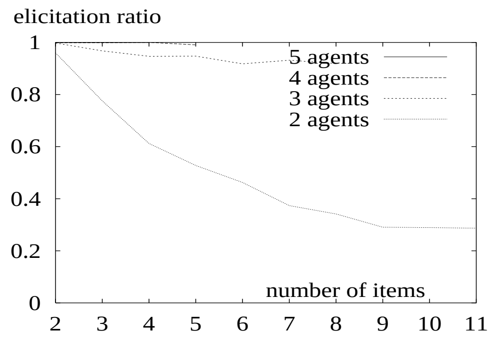

Figure 1: Performance of rank lattice based elicitation. The curves for 4 and 5 agents are barely visible, being at an elicitation ratio of almost 1. The experiment setup is described in the appendix.

4.1 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. Preliminary results indicate that the ratio continues to decrease as we run on instances with more than 11 items.

Unfortunately, Figure 1 also shows that as the number of agents n grows, the advantage from rank lattice based elicitation decreases. This phenomenon 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 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 algorithm

[2] A recent theoretical result shows that even with free disposal, *in the worst case*, finding an (even only approximately) optimal allocation requires exponential communication [Nisan and Segal, 2002].

[3] This was included in the *augmented order graph* of Conen & Sandholm [Conen and Sandholm, 2001].

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). These elicitors include all rank lattice based elicitors that search contiguous regions in the lattice, starting from the root (such as the EBF elicitor family studied in [Conen and Sandholm, 2002]).

5 General elicitation framework

Given that no rank lattice based elicitor can do much better than the one outlined above, we now move to a more general elicitation framework. As we will show, this allows us to develop algorithms that ask significantly fewer queries, and that scale well as the number of agents grows.

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 [Conen and Sandholm, 2001]:

```
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 $LB(c) = \sum_i LB_i(c_i)$ 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.

5.1 Determining domination

The PRUNE procedure needs to be able to determine whether a candidate c dominates another candidate c' – that is, whether the elicitor has enough information to prove that the value of c is at least as high as that of c'. Define $\delta(c, c')$ to be the least possible difference between $v(c)$ and $v(c')$ that is consistent with the information elicited so far. Clearly, if $\delta(c, c')$ is positive, then c has value at least as high as that of c' and thus c dominates c'. Similarly define $\delta_i(b, b')$ to be the least difference between $v_i(b)$ and $v_i(b')$ that is consistent. Namely, if $b \succeq b'$, then $\delta_i(b, b') = \max(0, LB_i(b) - UB_i(b'))$; otherwise, $\delta_i(b, b') = LB_i(b) - UB_i(b')$. Since there are no interactions between agents' private valuation functions, $\delta(c, c') = \sum_{i=1}^{n} \delta_i(c_i, c'_i)$.

Checking whether bundle c_i dominates bundle c'_i is an expensive operation: the elicitor needs to determine whether a path exists from c_i to c'_i in the constraint network. This takes time linear in the size of the network, which is $O(2^k)$ where k is the number of items.[4] Therefore, we show that if $UB(c) < UB(c')$ or $LB(c) < LB(c')$ then c does not dominate c'. Since these two relations can be checked in time independent of the number of items, this can save us time when the tests apply, which in practice they often do.

Proposition 1 *If $UB(a) < UB(b)$ then $a \not\succeq b$.*

Proof: Split the agents into two groups: $S = \{i | UB_i(a_i) < UB_i(b_i)\}$ and $\bar{S} = \{i | UB_i(a_i) \geq UB_i(b_i)\}$. $\bar{S}$ may be empty, but S necessarily includes at least one element by the assumption. For all $i \in S$, the propagation rules ensure that $a_i \not\succeq b_i$. This allows us to conclude that $\sum_{i \in S} \delta_i(a_i, b_i) = \sum_{i \in S} LB_i(a_i) - UB_i(b_i)$.

For $i \in \bar{S}$, $\delta_i(a_i, b_i)$ depends on whether $a_i \succeq b_i$. But if so, δ only increases. Thus

$$\begin{aligned}\sum_{i\in\bar{S}} \delta_i(a_i, b_i) &\leq \sum_{i\in\bar{S}} \max(0, LB_i(a_i) - UB_i(b_i)) \\ &\leq \sum_{i\in\bar{S}} \max(0, UB_i(a_i) - UB_i(b_i)) \\ &\leq \sum_{i\in\bar{S}} UB_i(a_i) - UB_i(b_i)\end{aligned}$$

The last step relies on the fact that by definition, for $i \in \bar{S}$, $UB_i(a_i) - UB_i(b_i) \geq 0$, so the max has no effect. Summing the δ_i over both sets, we get that $\delta(a, b) \leq UB(a) - UB(b)$. By assumption, $UB(a) - UB(b)$ is strictly negative. ■

Proposition 2 *If $LB(a) < LB(b)$ then $a \not\succeq b$.*

Proof: The proof is symmetric to the former proof.

Given these propositions, we implemented the following algorithm for testing domination:

```
DOMINATES$(c, c')$
1  if $UB(c) < UB(c')$ then return false
2  if $LB(c) < LB(c')$ then return false
3  if $\delta(c, c') < 0$ then return false
4  else return true
```

6 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 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 *In order to determine the 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.*

[4]Alternatively, one could keep in a hash table, for each bundle-agent pair (b, i), all the bundles that b dominates for that agent i. This would lead to constant time domination tests, but $O(2^{2^k})$ time for each edge addition and $\Theta(2^k)$ space for each bundle-agent pair, that is, space $\Theta(n2^{2^k})$.

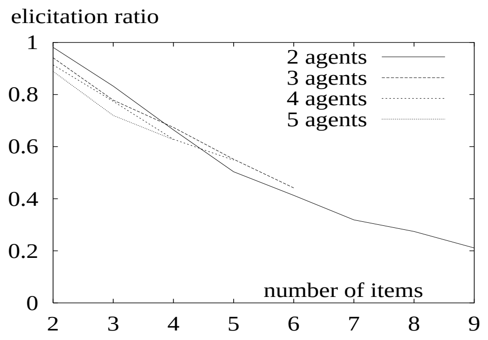

Figure 2: Elicitation using value and order queries.

Proof: The lower bound on the optimal allocation is finite (say, L) because we require each bundle to have non-negative and finite value for every bidder. Therefore, unless the auctioneer provides an upper bound on $v_i(K)$, the possibility is open that allocating K to i is worth more than L. Because allocating K to i possibly has value greater than implementing the allocation that is, in fact, optimal, the elicitation policy cannot terminate. ■

7 Order queries

In some applications, agents might not know the values of bundles, and might need to expend great effort to determine them [Parkes, 1999; Sandholm, 2000; Larson and Sandholm, 2001], 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 in general it cannot determine a social welfare maximizing allocation. However, order queries can be helpful when interleaved with other types of queries.

7.1 Interleaving value and order queries

We developed an elicitation policy that uses both value and order queries. It mixes them in a straightforward way, simply alternating between the two, starting with an order query. Whenever 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 a random tuple. Whenever a value query is to be asked, the elicitor chooses a random (b, i) where b is allocated to agent i in some candidate.

To evaluate the mixed policy, we need a way of comparing the cost of an order query to the cost of a value query. In the experiment, we let an order query cost 0.1 and a value query cost 1, capturing the notion that the qualitative order queries should be much easier to answer than precise value queries.

Figure 2 shows that, as desired, the elicitation ratio falls as the number of items increases. Furthermore, unlike with rank lattice based elicitors, this benefit is largely maintained as the number of agents increases.

The policy described above is able to reduce the number of precise values it elicits, by about 10%, over asking only value queries. This decrease is almost exactly offset by the cost of asking order queries. In other words, the order queries are helping, but more work needs to be done to find a policy that better combines the two query types—or at least, to find a better policy for 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 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 [Parkes, 1999; Sandholm, 2000; Larson and Sandholm, 2001]. For this reason, we introduce a new query type: a *bound-approximation query*. In such a query, the elicitor asks an agent i to tighten the agent's upper bound $UB_i(b)$ (or lower bound $LB_i(b)$) 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.

The elicitor can provide a hint t to the agent as to how much additional time the agent should devote to tightening the bound in the query. Smaller values of the hint t make elicitation more incremental, but cause additional communication overhead and computation by the elicitor. Therefore, the hint can be tailored to the setting, depending on the relative costs of communication, bundle evaluation by the bidders, and computation by the elicitor. The hint could also be adjusted at run-time, but in the experiments below, we use a fixed hint $t = 0.2$.

To evaluate this elicitation method, we need a model on how the agents' computation refines the bounds. We designed the details of our elicitation policy motivated by the following specific scenario, although the elicitation policy can be used generally. Let each agent have two anytime algorithms which it can run to discover its value of any given bundle: one gives a lower bound, the other gives an upper bound. Spending time d, $0 \leq d \leq 1$ will yield a lower bound $v_i(b)\sqrt{d}$ or an upper bound $(2 - \sqrt{d})v_i(b)$.[5] This means that there are diminishing returns to computation, as is the case with most anytime algorithms.[6] Finally, we assume that the algorithms can be restarted from the best solution found so far with no penalty: having spent d time tightening a bound, we can get the bound we would have gotten spending $d' > d$ by only spending an additional time $d' - d$.

[5]The model of agents' computation cost here opens the possibility to cheat in the evaluation of the elicitor. As the model is stated, the elicitor could ask an agent to spend t time each on the upper and lower bound. Based on the answers, the elicitor would know the exact value (it would be in the middle between the lower and upper bound). To check that our results do not inadvertently depend on such specifi cs of the agents' computation model, we ran experiments using an asymmetric cost function (linear for lower bounds, square root for upper bounds). This did not appreciably change the results.

[6]The square root is arbitrary, but captures the case of diminishing returns to additional computation. Running experiments with d in place of $\sqrt{d}$ did not signifi cantly change the results.

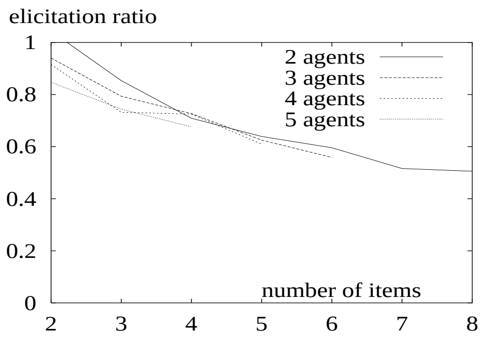

Figure 3: Elicitation using bound-approximation queries. The 2-agent, 2-item instances average an elicitation ratio greater than 1 because the method can incur up to cost 2 per bundle (1 each for tight upper and lower bounds)

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 $z = UB_i(b)$ while when computing the benefit of an upper query, $z = LB_i(b)$.[7] Computing the expected benefit rather than the optimistic benefit gave very similar results.

We evaluated bound-approximation queries using the elicitation policy and agents' computation model described above. Figure 3 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 Determining VCG payments

Having allocated the items to the agents, the auctioneer needs to specify how much each agent should pay for its bundle. Requiring an agent to pay the amount it revealed during the elicitation algorithm has the disadvantage that agents will be motivated to lie about their preferences (and may need to spend additional computational resources to compute what preferences they should reveal). In the Vickrey-Clarke-Groves (VCG) mechanism [Groves, 1973] applied to a combinatorial auction (this mechanism is also known as the *Generalized Vickrey Auction* [GVA]), the auctioneer charges each agent an amount equal to the negative externality that agent imposed on the other bidders. That is, if an agent i enters an auction and wins items, the other agents will typically be worse off than if the agent had not entered the auction; agent i is required to pay the difference to the auctioneer.

It has been proven that under the VCG pricing scheme, answering the elicitor's queries truthfully is an *ex post* equilibrium [Conen and Sandholm, 2001]. This is a weaker solution concept than the dominant strategy equilibrium of the full-revelation GVA, but still stronger than a Nash equilibrium. The difference is due to the fact that the elicitor's queries leak information to the bidder about what the other bidders have answered so far.

[7] A minor detail comes in estimating the worth of reducing an upper bound from ∞. We avoid this question by initially asking each agent for an upper bound on the grand bundle—which is almost always required anyway as shown in Proposition 3.

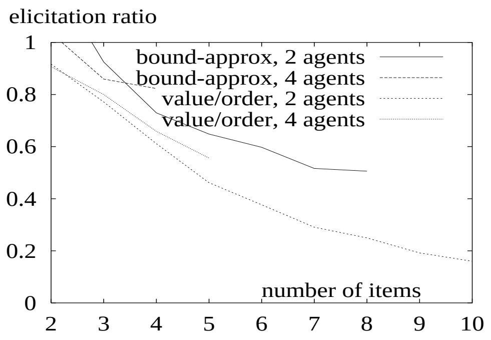

Figure 4: Computing VCG payments. The upper two curves are for the bound-approximation policy, the lower two for the value and order policy.

After enough information has been elicited to determine the optimal allocation, some additional elicitation may be required to determine the VCG payments. We implemented the following routine to carry out the overall elicitation:

```
COMPUTEPAYMENTS()
1  Call SOLVE as before, getting the optimal allocation $\mathrm{opt}$.
2  Elicit the exact value $v(\mathrm{opt})$ of the optimal allocation.
3  For each agent $i$, call SOLVE but remove from $C$ all allocations that allocate items to $i$. Call this $\mathrm{opt}_{-i}$. Elicit the exact value $v(\mathrm{opt}_{-i})$ of this allocation.
4  The payment by agent $i$ is $v(\mathrm{opt}_{-i}) - (v(\mathrm{opt}) - v_i(\mathrm{opt}_i))$.
```

In the 2-agent case, almost no additional elicitation is required: opt_{-i} simply allocates the grand bundle K to the agent that was not removed. Thus at most 4 additional values are needed over what is necessary to compute the optimal allocation: $v_1(\mathrm{opt}_1)$, $v_1(K)$, $v_2(\mathrm{opt}_2)$, and $v_2(K)$. While this argument does not generalize to more than 2 agents, in practice, the information needed for the VCG payments is elicited largely as a side effect of eliciting information for determining the optimal allocation. For example, the elicitation ratio of the bound-approximation policy is 60% at $n = 3$, $k = 5$ while computing VCG payments only increases the elicitation ratio to 71%. Similarly, that of the value and order policy only increases from 48% to 56%.

While the agents have no incentive to lie during COMPUTEPAYMENTS, they also have no incentive to tell the truth. In a more realistic setting, the elicitor should interleave the additional queries needed to compute the opt_{-i} allocations with the queries needed to compute opt.

10 Reverse auctions

While earlier work on preference elicitation has focused on combinatorial forward auctions, the methodology can be adapted for combinatorial *reverse* auctions as well, where there is one buyer and multiple sellers (bidders). For all the elicitation policies discussed in the general elicitation framework, the only change is in the PRUNE procedure. Rather than removing candidates that are dominated, we remove candidates that dominate.

Figure 5 shows some of the results of running our elicitors on combinatorial reverse auctions with bound-approximation queries (results for the value and order policy are qualitatively

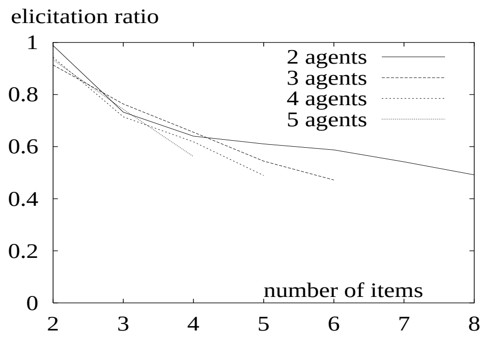

Figure 5: Bound-approximation queries in reverse auctions.

similar but omitted for lack of space). As happens with forward auctions, the elicitation ratio in reverse auctions falls as the number of items increases. Interestingly, the experiments indicate that while in auctions, adding more agents tends to increase the elicitation ratio, the converse is true in reverse auctions.

11 Conclusions and future work

In all of the elicitation algorithms of this paper, as the number of items for sale increases, the amount of information elicited is a small and shrinking fraction of the information collected in traditional "direct revelation mechanisms" where bidders reveal all their valuation information! With the exception of rank lattice based elicitors, the elicitation schemes largely maintain their benefit as the number of agents increases; in reverse auctions, in fact, the benefit grows with the number of agents.

By using the VCG pricing scheme, each agent is motivated to answer the queries truthfully, even if the agents are allowed to pass on queries and answer queries that were not asked. We showed that determining the VCG payments requires very little additional preference elicitation beyond what is needed to determine the optimal allocation.

We experimented with several query types. Using a combination of value queries to get exact values, and order queries which are easier to answer, we can reduce the amount of exact valuation the bidders need to do. More work needs to be done to reduce the overall amount of agent-side computation. Our bound-approximation queries take the incremental nature of elicitation to a new level. The agents are only asked for rough bounds on valuations first, and more refined approximations are elicited only on an as-needed basis. A related approach would be to propose a bound, and ask whether the agent's valuation is above or below the bound. This suggest a relationship between preference elicitation and ascending combinatorial auctions where the auction proceeds in rounds, and in each round the bidders react to price feedback from the auctioneer by revealing demand (e.g., [Parkes and Ungar, 2000; Wurman and Wellman, 2000]). As future research, we plan to explore this connection more deeply. We also desire to design new, increasingly effective preference elicitation algorithms.

Appendix: Experimental setup

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 each instance in under 2 minutes on a 2.8 GHz Intel machine.

Unfortunately, real data for combinatorial auctions are not publicly available. Therefore, as in all of the other academic work on combinatorial auctions 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 [Leyton-Brown *et al.*, 2000] as well as for many other prior benchmarks [Fujishima *et al.*, 1999; Sandholm, 2002a]. 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 honor the free disposal constraints.

The instances were generated by assigning, for each agent in turn, integer valuations using the following routine. 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 $G$
```

References

[Conen and Sandholm, 2001] Wolfram Conen and Tuomas Sandholm. Preference elicitation in combinatorial auctions: Extended abstract. *ACM-EC*. More detailed version: IJCAI-2001 Workshop on Economic Agents, Models, and Mechanisms.

[Conen and Sandholm, 2002] Wolfram Conen and Tuomas Sandholm. Partial-revelation VCG mechanism for combinatorial auctions. *AAAI*.

[Fujishima *et al.*, 1999] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. *IJCAI*.

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

[Hoos and Boutilier, 2001] Holger Hoos and Craig Boutilier. Bidding languages for combinatorial auctions. *IJCAI*.

[Larson and Sandholm, 2001] Kate Larson and Tuomas Sandholm. Costly valuation computation in auctions. *TARK*.

[Leyton-Brown *et al.*, 2000] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. *ACM-EC*.

[Nisan and Segal, 2002] Noam Nisan and Ilya Segal. The communication complexity of effi cient allocation problems. Draft. Second version March 5th, 2002.

[Nisan, 2000] Noam Nisan. Bidding and allocation in combinatorial auctions. *ACMEC*.

[Parkes and Ungar, 2000] David C Parkes and Lyle Ungar. Iterative combinatorial auctions: Theory and practice. *AAAI*.

[Parkes, 1999] David C Parkes. Optimal auction design for agents with hard valuation problems. *IJCAI-99 Agent-Mediated Electronic Commerce Workshop*.

[Rothkopf *et al.*, 1990] Michael H Rothkopf, Thomas J Teisberg, and Edward P Kahn. Why are Vickrey auctions rare? *Journal of Political Economy*, 98(1):94–109.

[Sandholm, 2000] Tuomas Sandholm. Issues in computational Vickrey auctions. *International Journal of Electronic Commerce*, 4(3):107–129. Early version: *ICMAS-96*.

[Sandholm, 2002a] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54. Early versions: *First International Conference on Information and Computation Economies*, 1998; Washington Univ., Dept. of Computer Science, tech report WUCS-99-01, 1999; *IJCAI-99*.

[Sandholm, 2002b] Tuomas Sandholm. eMediator: A next generation electronic commerce server. *Computational Intelligence* Special issue on Agent Technology for Electronic Commerce. (To appear). Early versions: AGENTS-00, AAAI-99 Workshop on AI in Electronic Commerce, Washington University, St. Louis, Dept. of Computer Science technical report WU-CS-99-02, 1999.

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