Generalizing Preference Elicitation in Combinatorial Auctions

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

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 [4], 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 vanishing 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., [1, 6, 11, 15, 19, 20]).

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 [9,13,18]; 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 [16]. Appropriate bidding languages [6, 8, 11, 17, 19] 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, and the latter problem leads to reduced economic efficiency of the resulting allocation of items to bidders.

Selective *incremental preference elicitation* by the auctioneer was recently proposed to address these problems [4], but the idea was not evaluated. In this paper we examine several different query types and elicitation policies for this purpose. We also study the complexity of not only determining the optimal allocation of items to bidders but also the bidders' Vickrey-Clarke-Groves (VCG) payments [3, 7, 21]. In addition to combinatorial auctions we study combinatorial reverse auctions (procurement auctions). Our experiments show that only a vanishing fraction of the bidders' preference information needs to be elicited in order to determine the provably optimal (economically efficient) 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

*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.

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] 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 [4, 5]: *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.

For example, consider two agents 1 and 2 bidding on two items A and B, and the following value functions:

$v_1(AB) = 8, \quad v_1(A) = 4, \quad v_1(B) = 3, \quad v_1(\emptyset) = 0$
$v_2(AB) = 9, \quad v_2(A) = 1, \quad v_2(B) = 6, \quad v_2(\emptyset) = 0$

So, agent 1 ranks AB first, A second, B third, and the empty bundle last. Agent 2 ranks AB first, B second, A third, and the empty bundle last.

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. For instance in the example above, rank vector $\langle 1, 2 \rangle$ corresponds to allocating AB to agent 1 and B to agent 2, which is infeasible. Similarly, rank vector $\langle 2, 2 \rangle$ allocates A to agent 1 and B to agent 2, which is a feasible allocation. The value of a rank vector r is $v(b(r)) = \sum_i v_i(b_i(r_i))$. Rank vector $\langle 1, 2 \rangle$ in our example has value $8 + 6 = 14$, while $\langle 2, 2 \rangle$ has value $4 + 6 = 10$.

The elicitor can put bounds on $v_i(b_i(r_i))$ using the constraint network as before. 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 $v_i(b_i(r_i))$. In our example, knowing only $b_1(1) = AB$ and $v_1(AB) = 8$, the elicitor can infer $v_1(b_1(2)) \leq 8$.

The set of all rank vectors defines a *rank lattice* (Figure 1). A key observation in the lattice is that the descendants of a node have lower (or equal) value to the node.

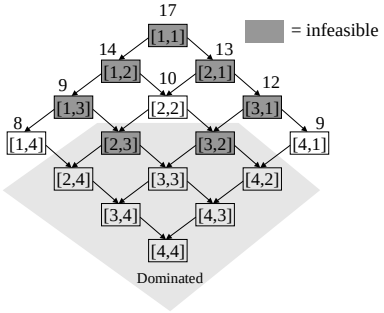

Figure 1: Rank lattice corresponding to the example. The gray nodes are infeasible. The shaded area is the set of nodes dominated by feasible nodes. The number above each node is the value of the node. At the outset, the auctioneer knows the structure of the lattice, but knows neither the shadings nor the values of each 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.

```
FindOptimal()
1  FRINGE ← {⟨1, 1, ... , 1⟩}
2  while FRINGE ≠ ∅
3     r = FindBestNode(FRINGE)
4     FRINGE ← FRINGE −{r}
5     if r is feasible
6        return r
7     for each r′ ∈ children(r)
```

[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.

[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 [12].

[3]This was included in the *augmented order graph* of Conen & Sandholm [4].

```
8       if r' ∉ FRINGE
9         FRINGE ← FRINGE ∪ {r'}
```

Unlike in typical best-first search, algorithm FindOptimal does not necessarily know which node of the fringe has highest value and thus should be expanded next. Determining this often requires 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 ← FRINGE
 2  remove from S all r dominated by some r' in S
 3  if all r ∈ S have the same value
 4    return arbitrary r ∈ S
 5  choose r ∈ 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
```

In some cases, FindBestNode can return a rank vector r although not all bundles $b_i(r_i)$ are known to the elicitor. This can occur, for example, if the known valuations in the rank vector already sum up to a large enough number. In that case, checking the feasibility in step 5 of FindOptimal requires eliciting the unknown bundles $b_i(r_i)$.

5. EXPERIMENTAL SETUP

While the idea and some algorithms for preference elicitation in combinatorial auctions have been presented previously [4], they have not been validated. To evaluate the usefulness of the idea, we conducted a host of experiments. We present the results in the rest of the paper. Each plot shows how many queries were needed to find an optimal allocation and prove that it is optimal (that no other allocation is better). 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 plots show results for the instance sizes on which all of the 50 instances took less than 2 minutes to solve on a 2.8 GHz Linux machine.

Unfortunately, real data for combinatorial auctions are not publicly available.[4] 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 [10] as well as for the other prior benchmarks [1, 6, 19]. In such cases, the communication is a non-issue, which undermines the purpose of elicitation. In addition, the instances generated by many of the earlier benchmarks do not honor the free disposal constraints (because for an agent, the value of a bundle can be less than that of a sub-bundle).

[4]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).

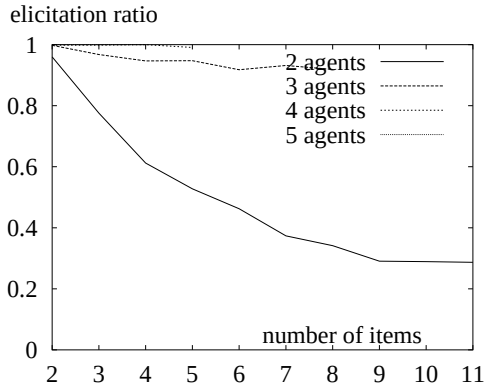

Figure 2: Performance of rank lattice based elicitation. The curves for 4 and 5 agents are barely visible, being at an elicitation ratio of almost 1.

In many real settings, each bidder has a nonzero valuation for every bundle. For example in spectrum auctions, each bidder has positive value for every bundle because each item is of positive value to every bidder (at least due to renting and reselling possibilities). In other settings, there may exist worthless items for some bidders. Even in such cases, under the free disposal assumption, the bidders have positive valuations for almost all bundles—except bundles that only contain worthless items (because, at worst, they can throw away the extra items in any bundle for free).

To capture these considerations, we developed a new benchmark instance 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)
 1  G ← new constraint network
 2  S ← 2^K (the set of all bundles)
 3  impose free disposal constraints on G
 4  UB(K) ← MAXBID
 5  while S ≠ ∅
 6    pick b uniformly at random from S
 7    S ← S − b
 8    pick v(b) uniformly at random from [LB(b), UB(b)]
 9    propagate LB(b) = UB(b) = v(b) through G
```

6. RANK LATTICE EXPERIMENTS

The first experiment evaluates 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 2 shows that as the number of items in the auction increases, the elicitation ratio approaches zero, that is, *only a vanishingly small fraction of the possible queries are asked!*

Unfortunately, Figure 2 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 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 continuous paths in the lattice, starting from the root (such as the EBF elicitor family studied in [5]).

7. 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 [4]:

```
SOLVE()
1  C ← INITIALCANDIDATES(n, k)
2  while not DONE(C)
3     q ← SELECTQUERY(C)
4     ASKQUERY(q)
5     C ← 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'). This may eliminate some optimal allocations, but it will never eliminate all optimal allocations—one will always remain. If strict domination were to be used as the criterion, then SOLVE would find *all* optimal allocations, at the cost of requiring more elicitation.

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.

7.1 Determining domination

As we stated, in the PRUNE procedure, 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'. 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 $b \succeq b'$, then $\delta_i(b, b') = \max(0, LB_i(b) - UB_i(b'))$; otherwise, $\delta_i(b, b') = LB_i(b) - UB_i(b')$. The elicitor can determine domination by computing $\delta(c, c')$.

The calculation of $\delta(c, c')$ requires checking whether bundle c_i dominates bundle c'_i. This 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 linear time in the size of the network, which is $O(2^k)$ where k is the number of items.[5]

It would be advantageous to avoid computing $\delta(c, c')$ exactly if possible. In this section we show how to accomplish this; by comparing the bounds on c and c', we can already rule out some domination relationships.

First, in some cases the elicitor can determine that there is no domination between *bundles* by simply looking at the bounds on the bundles' values. (This is not totally obvious. Consider the setting where the elicitor has been told that $a_i \succeq b_i$, $v_i(a_i) \in [0, 5]$, $v_i(b_i) \in [0, 5]$. In this setting the elicitor knows that a_i dominates b_i, although this cannot be inferred from the bounds.)

LEMMA 1. *If $UB_i(a_i) < UB_i(b_i)$ then $a_i \not\succeq b_i$.*

LEMMA 2. *If $LB_i(a_i) < LB_i(b_i)$ then $a_i \not\succeq b_i$.*

PROOF. Both of these Lemmas are immediate from the propagation rules. □

When the elicitor is determining domination between *candidates*, additional complications arise from the fact that different agents may rank the candidates in opposite order. Nevertheless, we show that in some cases the elicitor can determine that there is no domination by simply looking at the bounds on the candidates' values:

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 $i \in S$, we know that $a_i \not\succeq b_i$ by Lemma 1. This allows us to conclude:

$$\sum_{i \in S} \delta_i(a_i, b_i) = \sum_{i \in S} LB_i(a_i) - UB_i(b_i)$$

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, so an upper bound on δ assumes $a_i \succeq b_i$:

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

[5] 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 $O(2^{2k})$ time for each edge addition and $\Theta(2^k)$ space for each bundle-agent pair, that is, space $\Theta(n2^{2k})$.

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. This leaves us with:

$$\begin{aligned}\delta(a,b) &\leq \sum_{i \in S}[UB_i(a_i) - UB_i(b_i)] + \sum_{i \in \bar{S}}[UB_i(a_i) - UB_i(b_i)] \\ &= UB(a) - UB(b) < 0 \quad \square\end{aligned}$$

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**

8. THE GRAND BUNDLE IS (ALMOST) ALWAYS REVEALED

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. In this section we show that this indeed is a good idea.

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 K is* not *allocated.*

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. □

Furthermore, at least for value queries, eliciting the grand bundle value for *every* agent comes at no loss:

PROPOSITION 4. *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).*

PROOF. If the optimal allocation involves allocating items to at least two bidders, then we are not allocating the full bundle to any agent, so by the proposition above, the auctioneer must elicit $v_i(K)$ for every i.

Otherwise, the optimal allocation involves allocating all items to a single agent i. For all $j \neq i$, the proposition applies and therefore the auctioneer must elicit $v_j(K)$. What is left to prove is that the auctioneer is at least as well off also eliciting $v_i(K)$.

If all other bidders have zero value on all sub-bundles (that is, the full bundle has positive value to them, but anything less has zero value), then the auctioneer need only place a lower bound on $v_i(K)$ that is higher than any other bidder's, which it can do by eliciting $v_i(K)$.

If any other bidder j has nonzero value on some bundle $K-b$, the auctioneer needs to prove that $v_j(K-b)+v_i(b) \leq v_i(K)$. In other words, the auctioneer needs to provide a lower bound on $v_i(K)$ that is sufficiently greater than the upper bound on $v_i(b)$. Using value queries and assuming free disposal, the auctioneer can prove an upper bound on $v_i(b)$ by eliciting b or any super-bundle b'. Similarly, it can prove a lower bound on K by eliciting K or any sub-bundle. However, it cannot prove sufficiently tight bounds to separate the optimal allocation (of K to i) from the suboptimal one by eliciting a single bundle: by eliciting a single bundle, it would prove $UB_i(b) = LB_i(K)$. Given that it must elicit two bundles, it may as well elicit $v_i(K)$ to provide the lower bound on that value.

The argument in the paragraph above generalizes to settings where there are many bidders j who have nonzero value for several bundles $K-b$. The auctioneer must prove a sufficiently tight lower bound on K, and sufficiently tight upper bounds on each of the bundle-agent pairs (b, i). Since not all the elicitations that support sufficiently tight upper bounds on b's can also support a sufficiently tight lower bound on K, the auctioneer must elicit at least one other bundle to support that lower bound. There may be more than one choice for this; one possible choice is to elicit $v_i(K)$. □

We use these observations in many of the elicitor algorithms: they first consider the grand bundle. We will discuss the specific elicitation policies below.

9. ORDER QUERIES

In some applications, agents might not know the values of bundles, and might need to expend great effort to determine them [9, 13, 18], 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.

9.1 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.

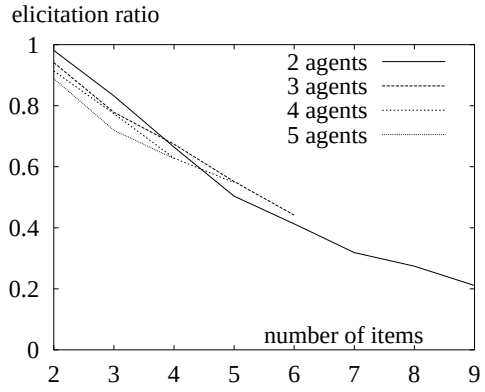

Figure 3: Elicitation using value and order queries.

Figure 3 shows that, as desired, the elicitation ratio approaches zero 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.

As the relative cost of an order query decreases, the benefit of interleaving value queries with order queries increases. In the limit, if order queries were free, the policy should ask all the order information at the outset, then ask value 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.

10. 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, 13, 18]. 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 $(2-\sqrt{d})v_i(b)$.[6] This means that there are diminishing returns to computation, as is the case with most anytime algorithms.[7] 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 $d' - d$.

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 expected benefit. This is the amount by which we expect the upper and lower bounds on bundle-agent pairs to be tightened when we propagate the new bound that the queried agent will return (only counting bundle-agent pairs that are included in the set of remaining candidates). To compute the expected benefit, the elicitor assumes that $v_i(b)$ is drawn uniformly at random in $[LB_i(b), UB_i(b)]$. To estimate the expected change in bounds, the elicitor samples 10 values $\hat{v}_i(b)$ in that interval, uniformly at random. For each value, the elicitor computes (using the cost model described in the previous paragraph) what bound z it would receive if the agent spent additional time t working on that bound and the true value were $\hat{v}_i(b)$. Finally, the elicitor observes by how much the values in the constraint network would change if the elicitor were to propagate z through the network (only bundle-agent pairs that are included in the set of remaining candidates are counted).[8]

We evaluated bound-approximation queries using the elicitation policy and agents' computation model described above. Figure 4 shows that as the number of items increases, only a vanishingly small fraction of the overall computation cost is actually incurred because 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. Because the

[6]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 specifics 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.

[7]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 significantly change the results.

[8]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. By free disposal, that is also an upper bound on all other bundles.

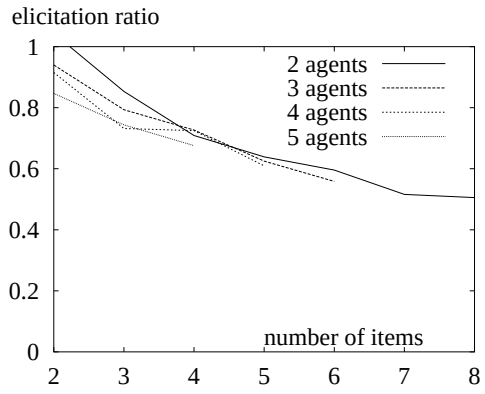

Figure 4: Elicitation using bound-approximation queries.

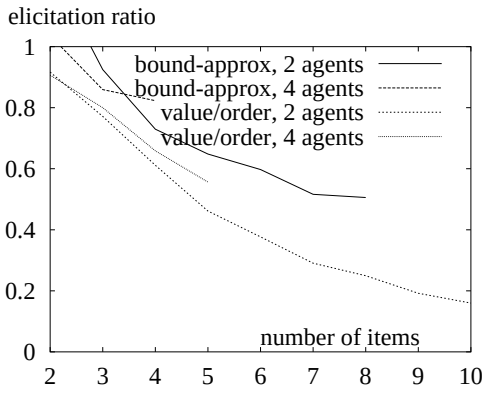

Figure 5: Computing VCG payments. The upper two curves are for the bound-approximation policy, the lower two for the value and order policy.

method can incur up to cost 2 per bundle (1 for getting an tight lower bound, plus another 1 for getting an tight upper bound), the elicitation ratio exceeds 1, as it does for the 2-agent, 2-item case.

11. 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 [3, 7, 21] applied to a combinatorial auction (this mechanism is also known as the *Generalized Vickrey auction*), the auctioneer charges each agents 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.

Under the VCG pricing scheme, answering the queries truthfully is an *ex post equilibrium* [4].[9] This holds despite the fact that the elicitor's queries leak information to the bidder about what the other bidders have answered so far. This also holds even if the bidders are allowed to pass on answering some queries, and are allowed to answer queries that were never asked!

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

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 precise values are needed over what is necessary to compute the optimal allocation: $v_i(\text{opt}_i)$ and $v_i(K)$ for both $i = 1$ and $i = 2$.Unfortunately this argument does not generalize to more than 2 agents. The optimal allocation and the VCG payments could be computed by invoking Solve separately $(n + 1)$ times (once for the overall problem, and once for each agent removed in turn). In practice, however, the information needed for the VCG payments is elicited largely as a side effect of eliciting information for determining the optimal allocation.[10] 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%.

12. 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.[11]

Figures 6 and 7 show some of the results of running our elicitors on combinatorial reverse auctions. As in auctions, only a vanishing fraction of the preferences of the bidders gets revealed. Interestingly, while in auctions, adding more agents tends to increase the elicitation ratio (for a sufficiently large number of items), *the converse is true in reverse auctions* (and "sufficiently large" is smaller).

13. CONCLUSIONS AND FUTURE WORK

In all of the elicitation algorithms of this paper, as the number of items for sale increases, *the amount of informa-*

[9]This is a game-theoretic solution concept that is stronger than Nash equilibrium, but weaker than dominant strategy implementation.

[10]In rank lattice based elicitors, *no* additional information is required to compute the VCG payments, regardless of the number of agents [5].

[11]For some policies, we might also need to redefine SelectOp, but the way the policies above choose an operation does not depend on whether the auction is a forward or a reverse auction.

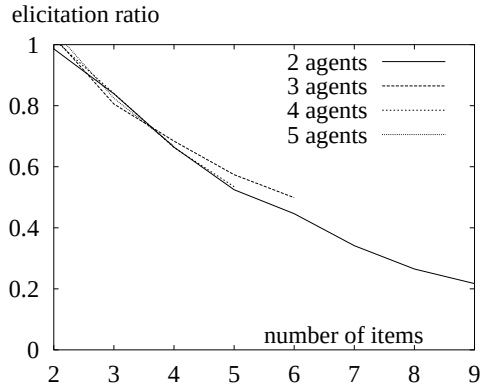

Figure 6: Alternating value/order query policy in reverse auctions.

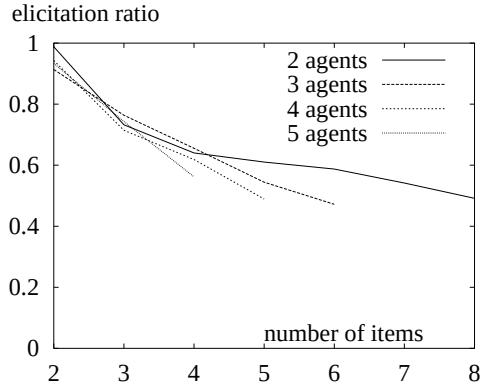

Figure 7: Bound-approximation query policy in reverse auctions.

tion elicited is a vanishing fraction of the information collected in traditional "direct revelation mechanisms" where bidders reveal all their valuation information! The elicitation schemes also maintain their benefit as the number of agents increases. Rank lattice based elicitors are the exception: we showed that they scale poorly in agents and we explained why.

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 adapted preference elicitation to combinatorial *reverse auctions* and showed that, there too, as the number of items for sale increases, the fraction of preference information that is revealed approaches zero. Unlike in auctions, the elicitation ratio *improves* as the number of agents increases.

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., [2,14,22]). As future research, we plan to explore this connection more deeply. We also desire to design new, increasingly effective preference elicitation algorithms.

14. REFERENCES

[1] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. *ICMAS* 2000.

[2] Sushil Bikhchandani, Sven de Vries, James Schummer, and Rakesh V. Vohra. Linear programming and Vickrey auctions, 2001. Draft.

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

[4] Wolfram Conen and Tuomas Sandholm. Preference elicitation in combinatorial auctions: Extended abstract. *ACM-EC* 2001.

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

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

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

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

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

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

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

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

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

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

[15] Michael H Rothkopf, Aleksandar Pekeč, and Ronald M Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

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

[17] Tuomas Sandholm. eMediator: A next generation electronic commerce server. *AGENTS* 2000.

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

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

[20] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for combinatorial auctions. *IJCAI* 2001.

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

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