Preference Elicitation in Combinatorial Auctions

Wolfram Conen
XONAR GmbH
Wodanstr. 7
42555 Velbert, Germany
E-mail: conen@gmx.de

Tuomas Sandholm*
Carnegie Mellon University
Computer Science Department
5000 Forbes Avenue
Pittsburgh, PA 15213
E-mail: sandholm@cs.cmu.edu

ABSTRACT

Combinatorial auctions (CAs) where bidders can bid on bundles of items can be very desirable market mechanisms when the items sold exhibit complementarity and/or substitutability. However, in a basic CA, the bidders may need to bid on exponentially many bundles. We present a topological structure inherent in the problem to reduce the amount of information that it needs from the bidders. An analysis tool is presented as well as data structures for storing and optimally assimilating the information received from the bidders. Using this information, the agent then narrows down the set of desirable allocations, and decides which questions to ask next. Several algorithms are presented that ask the bidders for value, order, and rank information. A method is presented for making the elicitor incentive compatible.

Tracking number: 563.
Content areas: Electronic commerce, markets, auctions, and exchanges; Bidding and bargaining agents; Conflicts, conflict resolution, and negotiation.

1. INTRODUCTION

Combinatorial auctions where bidders can bid on bundles of items can be desirable market mechanisms when the items exhibit complementarity or substitutability, so the bidder’s valuations for bundles are not additive. In this paper we present a design of an auctioneer agent that uses topological structure inherent in the problem to reduce the amount of information that it needs from the bidders. An analysis tool is presented as well as data structures for storing and optimally assimilating the information received from the bidders. Using this information, the agent then narrows down the set of desirable (welfare-maximizing or Pareto-efficient) allocations, and decides which questions to ask next. Several algorithms are presented that ask the bidders for value, order, and rank information. A method is presented for making the elicitor incentive compatible.

2. COMBINATORIAL AUCTION SETTING

In a combinatorial auction, the seller has a set \( \Omega = \{ \omega_1, \ldots, \omega_n \} \) of indivisible, distinguishable items that she can sell. Any subset of the items is called a bundle. Each bidder has a value function \( v_i : 2^{\Omega} \to \mathbb{R} \) that states the value that the bidder is willing to pay for any given bundle.\(^1\)

A collection \((X_1, \ldots, X_n)\) states which bundle \(X_i \subseteq \Omega\) is earmarked for each bidder \(i\). In a collection, some bidders’ bundles may overlap in items, which would make the collection infeasible. We call a collection an allocation if it is feasible, i.e., each item is allocated to at most one bidder. The possibility that \(X_i = \emptyset\) is allowed.

An allocation \(X\) is welfare maximizing if it maximizes \(\sum_{i=1}^{n} v_i(X_i)\) among all allocations (feasible collections). We call an allocation \(X\) Pareto efficient if there is no other allocation \(Y\) such that \(v_i(X_i) \geq v_i(Y_i)\) for each bidder \(i\) and strictly for at least some bidder \(i\).\(^3\)

In our model, the seller has zero reservation prices on all bundles, i.e., she gets no value from keeping them. If in reality she has reservation prices on bundles, that can be modeled by treating the seller as one of the bidders who submits bids that correspond to the reservation values.

\(^1\)In our model, the seller has zero reservation prices on all bundles, i.e., she gets no value from keeping them. If in reality she has reservation prices on bundles, that can be modeled by treating the seller as one of the bidders who submits bids that correspond to the reservation values.

\(^3\)Our definition of Pareto efficiency is based on compari-
Our definitions of desirability of the solution (welfare maximization and Pareto efficiency), are defined with respect to the reported valuations. Later in this paper we will present a method for motivating the agents to reveal their valuations truthfully within our elicitation method.

3. TOPOLOGICAL STRUCTURE IN COMBINATORIAL AUCTIONS

We observed significant topological structure in the combinatorial auction setting. We use that to avoid asking the bidders unnecessary questions about their valuations.

3.1 Rank Lattice and Associated Algorithms

Conceptually, the bundles can be ranked for each agent from most preferred to least preferred. This gives a unique rank for each bundle for each agent. The key observation behind the rank lattice is the following. Without referring to the values of the bundles, each collection can be mapped to a unique rank vector \([ R_1(X_1), R_2(X_2), \ldots, R_n(X_n) ]\). The set of rank vectors, and a dominates relation between them define a lattice. Now, the important fact is that if a collection (resp. its rank vector) is feasible (i.e., is an allocation), then no collection (resp. its rank vector) “below” it can be a better solution to the allocation problem.

Example 1. Let there be two goods, \( A \) and \( B \), and two agents, \( a_1 \), and \( a_2 \). The agents rank the bundles as follows.

Agent \( a_1 \): \((1 : AB, 2 : A, 3 : B, 4 : \emptyset)\)

Agent \( a_2 \): \((1 : AB, 2 : B, 3 : A, 4 : \emptyset)\)

This implies the rank lattice of Fig. 1. Only a subset of the collections is feasible and, thus, corresponds to allocations.

Figure 1: Left: Rank lattice for Example 1. The nodes are collections. Some of them are dominated, some are infeasible, some are both, and some are neither. Right: Rank lattice, augmented with sums of values for Example 2.

If a feasible collection is not dominated by another feasible collection, it is Pareto-efficient. In Figure 1, the set of Pareto-efficient solutions is \(\{[2,2], [1,4], [4,1]\}\).

Using the rank lattice as an analysis tool, we can now use search algorithms to find the optimal allocations! Specifically, we structure the problem as search, and as the search algorithm expands a search node, it asks the bidders queries on bundles within an agent. If payments are taken into account in the definition of Pareto efficiency, then the set of Pareto-efficient solutions collapses to equal the set of welfare-maximizing solution. We use the term welfare maximizing for the latter set, and reserve the term Pareto efficient for the former purpose.

3.1.1 Search Algorithm for Elicitation with Systematic Rank Queries

The following search algorithm that operates top-down along the implicitly given rank lattice finds all Pareto-efficient allocations. To generate successors in the lattice, it asks the agents what their next most preferred bundles are.

\(s = (1, \ldots, 1)/* \text{start node} */\)

\(\text{OPEN} = \{\}; /* \text{Unexpanded nodes} */\)

while \(\text{OPEN} \neq \{\} \) do

\( \text{Remove}(c, \text{OPEN}); \quad \text{SUC} = \text{suc}(c) \)

if Feasible(c) then

\( \text{PAR} = \text{PAR} \cup \{c\}; \quad \text{Remove}(\text{SUC}, \text{OPEN}) \)

else foreach \( n \in \text{SUC} \) do

if \( n \notin \text{OPEN} \) and Undominated(\( n, \text{PAR} \))

then Append(\( n, \text{OPEN} \))

In summary, the elicitation algorithm above keeps asking rank queries from the bidders until the algorithm has determined all Pareto efficient allocations in the auction. For each agent, the queries occur in the following order: what is your most preferred bundle, what is your second most preferred bundle, and so on.

Proposition 1. The algorithm above determines the set of Pareto-efficient solutions.

Proofs are omitted due to limited space.

3.1.2 Search Algorithm for Elicitation with Systematic Rank Queries and Value Queries

If (monetary) valuations can be asked, then value queries can be combined with rank queries to guide the search that finds a welfare-maximizing allocation.

Example 2. Let there be the two goods, \( A \) and \( B \), and the two agents, \( a_1 \), and \( a_2 \) of the above example. The agents assign the following values to the bundles:

\[
\begin{array}{c|ccc}
   & A & B & AB \\
\hline
a_1 & 0 & 4 & 3 & 8 \\
a_2 & 0 & 1 & 6 & 9 \\
\end{array}
\]

The values imply the preference order that has been considered in Example 1. The value-augmented rank lattice is shown in Figure 1 Right. The welfare-maximizing allocation is given by rank vector [2, 2], that is \(X^* = \{(A), (B)\}\).

The following search algorithm uses rank and value information to determine a welfare-maximizing allocation.

\(s = (1, \ldots, 1)/* \text{start node} */\)

\(\text{OPEN} = \{s\}; /* \text{Unexpanded nodes} */\)

\(\text{CLOSED} = \emptyset; /* \text{Expanded nodes} */\)

while \(\text{OPEN} \neq \{\} \) do

\( e = \arg \max_{c \in \text{OPEN}} \sum_{i \in N} v_i(c_i) \)

\(\text{OPEN} = \text{OPEN} \setminus \{c\}\)

if Feasible(c) then return (c)

\( \text{CLOSED} = \text{CLOSED} \cup \{c\}; \quad \text{SUC} = \text{suc}(c) \)

foreach \( n \in \text{SUC} \) do

if \( n \notin \text{OPEN} \) and \( n \notin \text{CLOSED} \)

then \( \text{OPEN} = \text{OPEN} \cup \{n\} \)
In practice, the algorithm would ask questions to determine the best rank vector in OPEN (i.e., to solve the arg max). For a given rank, the algorithm needs to know which bundle the agent associates with that rank and which value she attributes to that bundle. The algorithm traverses the rank lattice in a way that leads to a natural sequence of questions for the bidder: asking for the highest ranking bundle first, then proceeding to the next best bundle, and so on.

**Proposition 2.** The algorithm above finds a welfare-maximizing allocation.

If desired, all the welfare-maximizing allocations can be found by continuing the search until all the nodes on the open list have value strictly less than the optimal social welfare.

**Remark:** With additional knowledge about the valuation functions, the collections to be checked for feasibility can be restricted. For example, in many auctions there is free-disposal \( v_i(X) \leq v_i(Y) \) if \( X \subseteq Y \), i.e., the bidder can always throw away extra items for free). Free disposal combined with our assumption of unique valuations allows for the following type of pruning. It is never necessary to check the feasibility of collections that include one of the highest ranking bundles (a rank 1) and a non-empty bundle (the highest ranking bundle is always the bundle including all items, every such collection must be infeasible). Free disposal can be used to accomplish much more pruning in general. However, our techniques do not assume free disposal; they can be made to capitalize on it if it holds.

### 3.2 Policy-Independent Elicitation Algorithms

The algorithms presented so far are based on search, and the search strategy imposes constraints on the query policy (which question is asked as a function of answers received so far). We now present algorithms that avoid this weakness. Questions can be asked in any order that the auctioneer considers most efficient given the answers received so far. As in the search-based elicitation algorithms, no unnecessary (from the perspective of all the information known and derivable at that time) questions are asked.

Our elicitor policy-independent elicitation algorithms can ask any bidder the following types of queries:

- **Order queries:** Which bundle do you prefer, A or B?
- **Value queries:** What is your valuation for bundle A? The bidder can answer with bounds or an exact value.
- **Rank queries:** In your preferences, what is the rank of bundle A? Which bundle has rank \( \leq \) in your preferences? (Later we also discuss the more natural question: If you cannot get the bundles that you have declared most desirable so far, what is your most desired bundle among the remaining ones? These queries are identical to the systematic rank queries asked by the search-based elicitation algorithms presented earlier in this paper.)

Our policy-independent elicitation algorithms differ based on which combinations of these query types they use. They optimally assimilate the query results into a data structure which fully combines the results from the different types of queries.

#### 3.2.1 Augmented Order Graph

The elicitor algorithms use a data structure called an augmented order graph \( G \) to assimilate the answers. It includes a node for each (bidder, bundle) pair \((i, X)\). It includes a directed arc from node \((i, X)\) to node \((i, Y)\) (always nodes of the same bidder) whenever \( v_i(X) > v_i(Y) \). We call this a domination arc \( \prec \). The graph \( G \) also includes an upper bound \( UB \) and a lower bound \( LB \) for each node. Finally, it includes a rank \( R_i(X) \) for every node. Some of these variables may not have values. Initially, \( G \) includes no edges. The upper bounds are initialized to \( \infty \), and the lower bounds to 0 (in the free-disposal case) or to \(-\infty \) (in the general case). All of the rank information is initially missing. If there is free disposal, edges are added to the graph to represent this: \(((i, X), (i, Y)) \in \prec \) whenever \( Y \subseteq X \).

![Figure 2: Order graph, allocations, and how they relate to the rank lattice.](image-url)

Figure 2 shows the augmented order graph for our 2-agent, 2-good example at a stage where some of the information from the bidders has not yet been asked. In the upper right corner, two allocations and their relation to the nodes in the graph are shown. These allocations are connected to the corresponding feasible collections (allocations) in the rank lattice. The lower bound \( LB \) of a collection is the sum of the lower bounds of the bundles in that collection. Similarly, the upper bound \( UB \) of a collection is the sum of the upper bounds of the bundles in that collection. In the example, the allocations can be ordered due to the available rank information. The allocation \( \{(A), (B)\} \) dominates the other. The highlighted rank vector represents the welfare-maximizing allocation. This, however, cannot be determined yet due to lack of information.

Our algorithms use the augmented order graph as the basic analysis tool. As new information is obtained, it is incorporated into the augmented order graph. This may cause new arcs to be added, bounds to be updated, or rank information to be filled in. As a piece of information is obtained and incorporated, our algorithms fully propagate its implications. The process is monotonic in that new information allows us to make more specific inferences. Edges are never removed, upper bounds never increase, lower bounds never decrease, and rank information is never erased.
3.2.2 General Elicitation Algorithm Framework

In different settings, answering some of the query types can be more natural and easier than answering others. Therefore, we designed different algorithms that use different subsets of the query types listed above (and different query policies). The algorithms share the following general skeleton, but differ based on how the specific procedures in this skeleton work. An augmented order graph $G$ and an input set $Y$ are expected as input to the algorithms. The type of input set $Y$ depends on the specific algorithm.

Algorithm $Solve(Y, G)$:

**while not Done($Y, G$) do**
- $o = \text{SelectOp}(Y, G)$ /* Choose question */
- $I = \text{PerformOp}(o, N)$ /* Ask bidder */
- $G = \text{Propagate}(I, G)$ /* Update graph */
- $Y = \text{Candidates}(Y, G)$ /* Curtail the set of candidate allocations */

In addition to this general structure, the different policy-independent elicitation algorithms share the procedures for efficiently comparing two collections for dominance and for optimally propagating value information, rank information, and order information in the augmented order graph. Both will be explained below, starting with the comparison.

Given two collections, $a$ and $b$, and the augmented order graph $G$, the following procedure is used to check whether $a$ dominates $b$. This is determined using a combination of value and order information (queried and inferred).5

Function $Dominates(a, b, G)$:

- $O_{ab} = FALSE$ /* Flag for order domination */
- $C_{ab} = 0$ /* Amount of value domination */

**foreach** $i \in N$ do
- if $LB_i^a > UB_i^b$
  - then $O_{ab} = TRUE$
- else if $a_i > b_i$
  - then $O_{ab} = TRUE$
- else $C_{ab} = C_{ab} + (LB_i^a - UB_i^b)$

if $C_{ab} > 0$ or ($C_{ab} = 0$ and $O_{ab} = TRUE$)
  - then return TRUE else return FALSE

**Proposition 3.** Given that the augmented order graph is consistent (that is, order and value information are not contradictory), Dominates returns TRUE if and only if enough information has been queried to determine that $a$ dominates $b$. Otherwise, FALSE will be returned.

Next, we discuss the propagation of newly received information. If value or order information is inserted into a previously consistent graph $G$, values of upper bounds are propagated in the direction of the edges and lower bounds in the opposite direction. This propagation is done via depth-first-search in $G$, so the propagation time is $O(v + e)$, where $v$ is the number of bundles (number of nodes in $G$ that correspond to the agent whose values are getting updated), and $e$ is the number of edges between these nodes. The insertion of rank information is performed as a sequence of insertions of new edges that reflect the derivable order information.

5The Dominates procedure does not explicitly use rank information because the implications of the rank information will have already been propagated into the value information in the bounds and the order information.

**Case 1:** Inserting a new lower bound at node $k$:
Procedure PropLB($k, G$) /* $G$ contains the set of edges $\succ */

**Pre** $l \in Pre$ do
- if $LB_k > LB_i$ then $LB_i = LB_k$; PropLB($l, G$)

**Case 2:** Inserting a new upper bound at node $k$:
Procedure PropUB($k, G$) $Suc = \{l: (k, l) \in G\}$

**Pre** $l \in Suc$ do
- if $UB_k < UB_l$ then $UB_l = UB_k$; PropUB($l, G$)

**Case 3:** Inserting a new edge $k \succ l$:
Procedure InsertEdge($k, l, G$)
- if $LB_k < LB_l$ then $LB_k = LB_l$; PropLB($k, G$)
- if $UB_k < UB_l$ then $UB_l = UB_k$; PropUB($l, G$)

**Case 4:** Inserting an exact valuation for node $k$:
Procedure InsertValue($k, v, G$)
- $LB_k = v$; PropLB($k, G$); $UB_k = v$; PropUB($k, G$)

**Case 5:** Inserting a rank for node $k$:
Procedure InsertRank($n, r, G$)
- $(i, b) = n$; $K = \{(j, c) \in V : j = i\}$
- if $\exists k \in K$ with $R_i < R_k$ and $R_i \leq R_k l \in K$ with $R_i < R_n$
  - then InsertEdge($k, n, G$)
- if $\exists k \in K$ with $R_k < R_n$ and $R_k l \in K$ with $R_i > R_n$
  - then InsertEdge($n, k, G$)

Given a set of newly retrieved information, $I$, and the augmented order graph $G$, the following procedure will insert the information and propagate it.

Procedure Propagate($I, G$):

**foreach** $i \in I$ do
- switch $i$ /* Structural switch */
  - (node $k$, UB $b$):
    - if $UB_k > b$ then $UB_k = b$; PropUB($k, G$)
  - (node $k$, LB $b$):
    - if $LB_k < b$ then $LB_k = b$; PropLB($k, G$)
  - (node $k$, node $l$): InsertEdge($k, l, G$)
  - (node $k$, value $v$): InsertValue($k, v, G$)
  - (node $k$, rank $r$): InsertRank($k, r, G$)

Based on the general structure and the procedures that have been discussed above, policy-independent algorithms can be derived that differ on the types of information that they request from the bidders. To obtain the complete algorithm, the procedures that we describe below for each different elicitation algorithm are plugged into the general structure above.

3.2.3 Algorithms that Query Value Information

Here we present the ingredients for an elicitation algorithm that determines the set of welfare-maximizing solutions using value queries. The following functions are plugged into the general structure that was described above.

First, we present the function Done$\text{es}$. Given a non-empty set of allocations $Y$, and an augmented order graph $G$, it checks whether all the allocations in $Y$ have the same value.
Function Done, (Y, G):
if |Y| = 1 then return TRUE
foreach a ∈ Y do
    lb = \sum_{n\in G_a} LB_n
    ub = \sum_{n\in G_a} UB_n
    if lb ≠ ub then return FALSE
return TRUE

PROPOSITION 4. The function Done, returns TRUE if and only if all allocations in Y have the same value. Otherwise it returns FALSE.

The function Candidates, determines from a set Y of allocations the subset that contains all allocations that are not known to be dominated, given the information available in G. The resulting set will only contain allocations that are pairwise incomparable with respect to the Dominates function.

Function Candidates, (Y, G):
O = ∅; C = ∅
while Y ≠ ∅ do
    pick a ∈ Y /* arbitrarily selects an element */
    Y = Y \ {a}; C = ∅
    while Y ≠ ∅ do
        pick b ∈ Y; Y = Y \ {b}
        if Dominates(b, a, G)
            then a = b
        else if not Dominates(a, b, G)
            then C = C \ {b}
        Y = C; O = O ∪ {a}
    return O

PROPOSITION 5. The function Candidates, determines the (maximal) set O of allocations such that for each allocation a in O there does not exist an allocation b in the input set Y which dominates a.

To complete the algorithm, a query policy to instantiate SelectOp needs to be specified. With the observation that a completely augmented order graph\(^8\) suffices to precisely decide all Dominates questions (and with the assumption that the information space is finite), any query policy that continuously adds new information to the graph G (until G is complete) can be used. This does not imply that the order graph must always be completely augmented to determine the solution set.

We propose the following simple policy as an example. Pick a node \(a = (i_1, b_1)\) from the set of not completely augmented nodes in G (that is, \(LB_a \neq GB_a\)) such that \(a\) is among the nodes in this set with the largest number of relations to allocations in Y. Ask agent \(i\) for the value of bundle \(b\). This simple policy directly asks for precise valuations. It adds new information to the graph in each round until the graph is completely augmented, which is sufficient for correctness of the algorithm. A more sophisticated query policy could ask upper and lower bounds on valuations. For example, the elicitor could, in effect, ask for incremental bidding against competitors’ bids.

PROPOSITION 6. Given a set of allocations Y and a graph G, the appropriately instantiated algorithm Solves, will determine the set of undominated allocations contained in Y.

If Y is initialized to the set of all allocations,\(^7\) the result will be the set of welfare-maximizing allocations.

If the set of welfare-maximizing allocations contains more than one element, all valuations of nodes that are part of those allocations have to be known—Done, will not terminate earlier. Could the function Done, be written more intelligently to avoid this? The answer is no because the Dominates function is the provably best device for evaluating the available information, and, after the first round, each set Y only contains allocations that are pairwise incomparable with respect to the Dominates function. Therefore, upon entering Done, we know that the pairwise incomparability is either due to the fact that information is missing or that the allocations have the same value—but only the latter is a correct reason to terminate. Thus, as long as we do not know whether the allocations in Y have the same value, additional information has to be requested.

In the following sections we present algorithms that query other information besides value information, and in those cases the social welfare maximizing set can often be determined without knowing the values of all allocations.

3.2.4 Algorithms that Query Order Information

Order information allows the Pareto-efficient allocations to be determined, but it cannot be used to determine welfare-maximizing allocations because that would require quantitative tradeoffs across agents.

Given a non-empty set Y of allocations and a graph G, the function Candidates, presented earlier, can be used to determine the set of allocations that are not dominated, given the information in hand.

Given a set of allocations U and the graph G, the following function will return FALSE if a pair of allocations exists in U which have been judged incomparable due to lack of information.

Function Done, (U, G)
foreach \{(a, b) ∈ U × U, a ≠ b\} do
    if not DefinitelyIncomparable\(^8\) (a, b)
        then if \(\exists i \in N : (i, a_i), (i, b_i) \not\succ\succ (i, a_{i'}) \not\succ\succ (i, b_{i'})\)
            and \((i, b_i), (i, a_i) \not\succ\succ\)
                then return FALSE
        return TRUE

The remaining part of the algorithm is the query policy. Our method can accommodate any policy here, but we suggest two intuitive ones as examples:

(1) Arbitrarily pick a pair of distinct allocations \(\{a, b\}\) that are incomparable due to a lack of information. Arbitrarily, choose one of the agents \(i \in N\), for which no order information for the corresponding bundles \(a_i\) and \(b_i\) is available. Ask the bidder which one of them she prefers.\(^9\)

---

\(^7\)This set can be determined from the possible partitions of \(\Omega\) and the possible permutations of the agents.

\(^8\)A pair \(\{a, b\}\) of allocations will be called definitely incomparable, if there is a pair of agents, \(\{i_1, i_2\}\) such that edges \((i_1, a_{i_1}) \succ (i_1, b_{i_1})\) and \((i_2, b_{i_2}) \succ (i_2, a_{i_2})\) and no edges \((i_1, b_{i_1}) \succ (i_1, a_{i_1})\) or \((i_2, a_{i_2}) \succ (i_2, b_{i_2})\) exist.

\(^9\)The answer to this question alone might not be sufficient to order \(a\) and \(b\) since there may be other unordered bundles in those allocations. Also, this question might not be necessary: it can be possible to deduce the answer from answers to other alternative questions. On the positive side, the an-
(2) Determine the set of pairs of incomparable allocations, \( U \). While doing so, determine a set \( P \) of pairs of unordered nodes \( \{(i, a_i), (i, b_i)\} \in G \) for which \( \exists a, b \in U, a \neq b \) so that neither \( (i, a_i), (i, b_i) \) \( \notin \) nor \( (i, b_i), (i, a_i) \) \( \notin \). Select from \( P \) a pair \( p = \{(i, b_1), (i, b_2)\} \) of nodes so that the number of pairs in \( U \) which contain \( p \) is maximal.\(^{10}\) Ask the bidder which bundle she prefers more, \( b_1 \) or \( b_2 \).

**Proposition 7.** Given a set of allocations, \( \mathcal{Y} \), and the graph \( G \), for either query policy, \( \text{Solve}_v(\mathcal{Y}, G) \) will determine the set of Pareto-efficient allocations contained in \( \mathcal{Y} \).

### 3.2.5 Algorithms that Query Value and Order Information

The algorithms for dealing with value information and for dealing with order information, described above, can be easily integrated to deal with both value and order information together. This is because in the \( \text{Candidates} \) function both algorithms already use both value and order information to the fullest. The order edges from the graph \( G \), and the value information, are used to determine which allocations are still undominated. Also, the order information is used to propagate upper and lower bound information across nodes in \( G \).

If the value-based algorithm’s \( \text{Done} \) function allows stopping, the set of welfare-maximizing allocations has been found. If the order-based algorithm’s \( \text{Done} \) function allows stopping, only the Pareto-optimal allocations have been determined so far.

Any query policy would suffice as long as it asks an agent order questions about the agent’s unordered bundles (that are included in currently undominated allocations), or value questions about bundles for which the upper and lower bound differ. The algorithm will then find the welfare-maximizing allocations. Interestingly, this generally does not even require knowing the value of those allocations because order information can substitute for detailed value information!

### 3.2.6 Algorithms that Query Rank Information

Algorithms that use rank information only cannot determine welfare-maximizing solutions because that requires quantitative tradeoffs across agents. Consequently, we present an algorithm for finding Pareto-efficient ones.

Given \( G \) and a set \( C \) of rank vectors, the following function answers \( \text{TRUE} \) if all elements of \( C \) are feasible.

Function \( \text{Done}_v(C, G) \)

**foreach** \( c \in C \) do

if \( \exists i \in N : \hat{\beta}(i, b) \in V \) with \( R_{(i, b)} = c_i \)

then return \( \text{FALSE} /* \text{Information missing} */ \)

if not Feasible\(^{11}\)(\( c, G \))

then return \( \text{FALSE} \) else return \( \text{TRUE} \)

Whenever an infeasible rank vector \( c \) is found, the children of \( c \) are generated (see \( \text{succ-function below} \)) and are inserted into \( C \) if and only if they are not dominated by one of the feasible or undominated rank vectors. Note that domination can be determined without knowing which actual bundle will correspond to a given rank.

**Proposition 8.** Given a set of rank vectors, \( C \), and the graph \( G \), for both policies, algorithm \( \text{Solve}_r(C, G) \) will determine the set of feasible rank vectors in the (partial) lattice determined by \( C \) that are either in \( C \) or dominated only by infeasible rank vectors in \( C \). If \( C \) is initialized to \( \{1, \ldots, 1\} \), the resulting set represents the set of Pareto-efficient allocations.

### 3.2.7 Algorithms that Query Rank and Value Information

In this section we show how rank and value information can be used to determine welfare-maximizing solutions. The \( \text{Candidates} \) and \( \text{Done} \) functions are instantiated as follows to accomplish this.

Function \( \text{Candidates}_r(C, G) \):

\[ c = \text{arg max}_{c \in C} \text{LB}^{12}(d, G) \]

if \( \exists d \in C \setminus \{c\} \) with \( \text{UB}(d, G) > \text{LB}(c, G) \)

then \( C = \text{Expand}(c, C, G) \)

**Done\(_r\)** checks whether the node in \( C \) with the greatest lower bound might be dominated by an as yet incomparable and potentially feasible rank vector. If so, \( \text{FALSE} \) is returned.

Function \( \text{Done}_v(C, G) \):

\[ c = \text{arg max}_{c \in C} \text{LB}(d, G) /* \text{Best valued node} */ \]

if \( \exists d \in C \setminus \{c\} \) with \( \text{UB}(d, G) > \text{LB}(c, G) \)

and not Feasible\(^{13}\)(\( d, G \))

then return \( \text{FALSE} \) else return \( \text{TRUE} \)

\(^{10}\)Deciding this edge adds information to the largest number of decisions in the next stage.

\(^{11}\)Feasibility can often be determined without knowing all rank-bundle relations. If the partial information available is not sufficient to decide about infeasibility, \( \text{FALSE} \) will be returned. Thus, if both functions return \( \text{FALSE} \), the information is insufficient.

\(^{12}\)LB\(_d\) = \( \sum_{i \in N} B(i, d_i) \), where \( B(i, d_i) \) is \( \text{LB}(i, b) \) if \( \exists (i, b) \in V \) with \( R_{(i, b)} = d_i \), and 0 otherwise. UB\(_d\) is determined analogously, with \( \infty \) in place of 0.
We propose the following query policy for $\text{SelectOp}_{\psi}(C, G)$. Pick the rank vector with the highest lower bound, say $c$ (c has some chance of being among the welfare-maximizing allocations). Pick from the remaining rank vectors a rank vector $d$ with $UB_d > LB_c$ ($d$ might end up being better than $c$ once enough information is available to decide the *dominates* relation between $c$ and $d$). Now, additional information will be requested. If there is a rank $r$ in position $i$ in $d$ without a related node (that is, neither the bundle that is ranked by agent $i$ at rank $r$ nor its valuation are known), ask agent $i$ which bundle she ranks at rank $r$ (this will help to determine the feasibility of the rank vector $d$). If no such position $i$ with missing bundle information exists, look for a position with no precise valuation information, that is, if there is a rank $r$ in a position $i$ of $d$ and a corresponding node $(i, b)$ with $UB(i, b) \neq LB(i, b)$, ask agent $i$ her value for bundle $b$ (this helps to improve the accuracy of the bounds on the overall valuation of $d$). Such a position $i$ will always exist because otherwise the valuation for $d$ would be precisely known already ($UB_d = LB_d$), and, with $UB_d > LB_c$, $LB_d$ would be greater than $LB_c$, which contradicts the selection of $c$.

**Proposition 9.** Given a set of rank vectors, $C$, and the graph $G$, algorithm $\text{Solve}_{\psi}(C, G)$ will determine the set of feasible rank vectors in the (partial) lattice determined by $C$ that are not dominated by other feasible rank vectors in the sublattice. If $C$ is initialized to $(1, \ldots, 1)$, the resulting set represents the welfare-maximizing allocations.

### 3.2.8 Algorithms that Query Value, Order, and Rank Information

An elicitor that uses value, order, and rank queries can also be instantiated into our general policy-independent elicitation framework. We omit the details of the instantiation. They are obvious in light of the functions presented for the algorithms that use a subset of these query types.

## 4. EFFICIENCY OF ELICITATION

The elicitor is economically efficient. Our algorithms that use value queries (possibly with other queries) are guaranteed to find the social welfare maximizing allocations. Our algorithms that only use order and/or rank queries are guaranteed to find the Pareto optimal allocations.

It is easy to show that the elicitor also saves revelation, which was the motivation for building an elicitor in the first place. Consider the following simple example. A bidder has revealed that she prefers bundle $A$ over bundle $B$, and that her valuation for $A$ is at most $100$. If, based on the bids from others, the elicitor already knows that it can obtain revenue higher than $100$ for bundle $B$, then the elicitor need not ask the bidder her valuation for $B$, because she would not win $B$ anyway.

It is also easy to show that in the worst case, the number of queries the elicitor needs to ask to determine the optimal allocation is exponential in the number of items for sale (at least when it comes to value and order queries). Consider the following simple example with just one bidder. Say the bidder assigns a high value to some bundle, and zero value to all other bundles. The elicitor’s goal of maximizing welfare amounts to trying to find the bundle that the bidder most prefers (and to prove that that bundle is better than any other). So, without any extra structure (such as knowledge of free disposal), the elicitor needs to get some information about the value of every bundle. Every value query provides information about only one bundle, and every order query provides information about only two bundles. Therefore, the number of value/order queries needed is at least half the number of bundles, which is exponential in items. Furthermore, a recent result by Nisan shows that no matter what type of information is communicated in a combinatorial auction, an exponential number of bits needs to be communicated in the worst case if the only extra knowledge is that of free disposal [6]. While the worst-case communication complexity is exponential, it may still be a vanishingly small fraction of all the valuation information possessed by the bidders. Also, one could strive to improve the worst-case communication complexity if the bidders’ valuation functions have special structure. Finally, one could strive to design elicitor algorithms that require low communication on average.

To improve the (average case) revelation efficiency, our elicitor can allow the bidders to also answer queries that were not asked (and our answer assimilation algorithms would not treat the answer any differently). This allows the bidder, who has some information about his own valuations that the elicitor does not have, to guide the revelation process. For example, this would solve the example of the previous paragraph. On the other hand, the elicitor also has information that the bidder does not have (about the others’ valuations) so in some cases the query-directed revelation is effective—as shown in the paragraph before last. Our elicitor supports both directions of guiding revelation, leading to a hybrid push-pull mechanism.

We can also integrate the elicitation technique with open-cry ascending combinatorial auctions, where some unnecessary revelation is avoided via price feedback [10, 9, 7, 18, 8]. Namely, if the price of a bundle is already too high for an agent, the agent need not compute or communicate her exact valuation. On top of that, the elicitor can guide revelation, and bidders can answer queries that were not asked.

## 5. INCENTIVE COMPATIBLE ELICITATION

Motivating the bidders to answer queries truthfully is a key issue, and is exacerbated by the fact that the elicitor’s queries may leak information to the bidder about the answers that other bidders have given.

However, any of our elicitor designs can be made incentive compatible in the sense that every bidder answering the queries truthfully is a perfect Bayesian equilibrium. This is accomplished by organizing the mechanism so that if all the bidders answer truthfully, the final allocation and payments follow the Vickrey-Clarke-Groves scheme (VCG) [17, 1, 3]. In the VCG, the amount a bidder has to pay is the sum of others’ revealed valuations for the bundles they get had the bidder not participated minus the sum of others’ revealed valuations for the bundles they get in the actual allocation.

The elicitor can determine these payments by asking enough queries to be able to determine the welfare maximizing allocation overall, and by asking extra queries to determine the welfare maximizing allocation for the auctions where each...
agent is ignored in turn. Conceptually, one could think of n+1 “elicitors”, each working to solve one of these problems. However, these “elicitors” can use the same data structure for assimilating the results, which leads to the advantage that queries answered for one “elicitor” can help another “elicitor” on its problem. Once all of the “elicitors” have found their welfare maximizing allocations respectively, the process terminates. The notion of multiple “elicitors” is just for conceptual clarity of this presentation; in practice there would be only one elicitor asking all of the queries.

There is the risk of a lazy bidder who would not answer queries once enough have been answered to determine her allocation and payment. To deter this possibility, the mechanism could interleave, for that bidder, the questions pertinent to that bidder’s allocation and VCG payment with queries pertinent to the other agents’ allocations and VCG payments. This way the bidder would not know (at least not directly) which purpose the questions are for. Any order would motivate the bidder to reveal truthfully—the interleaving scheme is simply to avoid laziness.

6. CONCLUSIONS AND FUTURE RESEARCH

Combinatorial auctions require potentially every bundle to be bid on, and there are 2^m − 1 bundles. This is complex for the bidder because she may need to invest effort or computation into determining each of her valuations. If the bidder evaluates bundles that she does not win, evaluation effort is wasted. Bidding on too many bundles can also be undesirable from the perspective of revealing unnecessary private information and from the perspective of causing unnecessary communication overhead. If the bidder omits evaluating (or bidding on) some bundles on which she would have been competitive, economic efficiency and revenue are generally compromised. A bidder could try to evaluate (more accurately) only those bundles on which she would be competitive. However, in general it is difficult for the bidder to know on which bundles she would be competitive before evaluating the bundles.

To address these problems, we presented a design of an elicitor that helps guide the revelation of information from the bidders to the auctioneer by asking relevant questions from the bidders and optimally assimilating the answers. It allows bidders to also answer queries that were not asked, leading to a push-pull hybrid mechanism for preference revelation. It can be used in conjunction with price-feedback mechanisms to get the best of all of the (known) mechanisms for guiding revelation. We also presented a way to make the elicitor incentive compatible.

Future research includes evaluating the savings in revelation both theoretically and experimentally, for different bid distributions with and without special structure. Future work also includes designing different query policies within our elicitor algorithm to bid distributions with different characteristics.

7. REFERENCES