An Algorithm for Multi-Unit Combinatorial Auctions

Kevin Leyton-Brown and **Yoav Shoham** and **Moshe Tennenholtz**
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

We present a novel algorithm for computing the optimal winning bids in a combinatorial auction (CA), that is, an auction in which bidders bid for bundles of goods. All previously published algorithms are limited to single-unit CAs, already a hard computational problem. In contrast, here we address the more general problem in which each good may have multiple units, and each bid specifies an unrestricted number of units desired from each good. We prove the correctness of our branch-and-bound algorithm, which incorporates a specialized dynamic programming procedure. We then provide very encouraging initial experimental results from an implemented version of the algorithm.

Introduction

Auctions are the most widely studied mechanism in the mechanism design literature in economics and game theory (Fudenberg & Tirole 1991). This is due to the fact that auctions are basic protocols, serving as the building blocks of more elaborated mechanisms. Given the wide popularity of auctions on the Internet and the emergence of electronic commerce, where auctions serve as the most popular game-theoretic mechanism, efficient auction design has become a subject of considerable importance for researchers in multi-agent systems (e.g. (Wellman *et al.* 1998; Monderer & Tennenholtz 2000)). Of particular interest are multi-object auctions where the bids name bundles of goods, called combinatorial auctions (CA). For example, imagine an auction of used electronic equipment. A bidder may wish to bid x for a particular TV and y for a particular VCR, but $z \neq x + y$ for the pair. In this example all the goods at auction are different, so we call the auction a single-unit CA. In contrast, consider an electronics manufacturer auctioning 100 identical TVs and 100 identical VCRs. A retailer who wants to buy 70 TVs and 30 VCRs would be indifferent between all bundles having 70 TVs and 30 VCRs. Rather than having to bid on each of the $\binom{100}{70} \cdot \binom{100}{30}$ distinct bundles, she would prefer to place the single bid (price, $\{$70 TVs, 30 VCRs$\}$). We call an auction that allows such a bid a multi-unit CA.

In a combinatorial auction, a seller is faced with a set of price offers for various bundles of goods, and his aim is to allocate the goods in a way that maximizes his revenue. This optimization problem is intractable in the general case, even when each good has only a single unit (Rothkopf, Pekec, & Harstad 1998). Given this computational obstacle, two parallel lines of research have evolved. The first exposes tractable sub-cases of the combinatorial auctions problem. Most of this work has concentrated on identifying bidding restrictions that entail tractable optimization; see (Rothkopf, Pekec, & Harstad 1998; Nisan 1999; Tennenholtz 2000; Vries & Vohra 2000). Also, the case of infinitely divisible goods may be tractably solved by linear programming techniques. The other line of research addresses general combinatorial auctions. Although this is a class of intractable problems, in practice it is possible to address interestingly-large datasets with heuristic methods. It is desirable to do so because many economic situations are best modeled by a general CA, and bidders' strategic behavior is highly sensitive both to changes in the auction mechanism and to approximation of the optimal allocation (Nisan & Ronen 2000). Previous research on the optimization of general CA problems has focused exclusively on the simpler single-unit CA (Fujishima, Leyton-Brown, & Shoham 1999; Sandholm 1999; Lehmann, O'Callaghan, & Shoham 1999)). The general multi-unit problem has not previously been studied, nor have any heuristics for its solution been introduced.

In this paper we present a novel algorithm, termed CAMUS (Combinatorial Auction Multi-Unit Search), to compute the winners in a general, multi-unit combinatorial auction. A generalization and extension of our CASS algorithm for winner determination in single-unit CA's (Fujishima, Leyton-Brown, & Shoham 1999), CAMUS introduces a novel branch-and-bound technique that makes use of several additional procedures. A crucial component of any such technique is a function for computing upper bounds on the optimal outcome. We present such an upper bound function, tailored specifically to the multi-unit combinatorial auctions problem. We prove that this function gives an upper bound on the optimal revenue, which enables us to show that CAMUS is guaranteed to find optimal allocations. We also introduce dynamic programming techniques to more efficiently handle multi-unit single-good bids. In addition, we present techniques for pre-processing and caching, and heuristics for determining search orderings, further capitalizing on the inherent structure of multi-unit combinatorial auctions.

In the next section we formally define the general multi-unit combinatorial auction problem. In Section 3 we describe CAMUS. In Section 4 we deal in some more detail with some of CAMUS's techniques. Due to lack of space, we cannot present all the CAMUS procedures in detail; however, this section will clarify its most fundamental components. In Section 5 we present our experimental setup and some experimental results.

Problem Definition

We now define the computational problem associated with multi-unit combinatorial auctions.

Let $G = \{g_1, g_2, \ldots, g_m\}$ be a set of goods. Let $q(j)$ denote the number of available units of good j. Consider a set of bids $B = \{b_1, \ldots, b_n\}$. Bid b_i is a pair $(p(b_i), e(b_i))$ where $p(b_i)$ is the price offer of bid b_i, and $e(b_i) = (e(b_i)_1, e(b_i)_2, \ldots, e(b_i)_m)$ where $e(b_i)_j$ is the number of requested units of good j in b_i. If there is no bid requesting k units of good i and 0 units of all goods $j \neq i$ (for some $1 \leq i \leq m$ and some $1 \leq k \leq q(i)$) then, w.l.o.g, we augment B with a bid of price 0 for that bundle. An allocation $\pi \subseteq B$ is a subset of the bids where $\Sigma_{b\in\pi} e(b)_j \leq q(j)$ $(1 \leq j \leq m)$. A partial allocation $\pi_{partial}$ is an allocation where, for some j, $\Sigma_{b\in\pi_{partial}} e(b)_j < q(j)$. A full allocation is an allocation that is not partial. Let Π denote the set of all allocations. The multi-unit combinatorial auction problem is the computation of an optimal allocation, that is, $argmax_{\pi\in\Pi} \Sigma_{b\in\pi} p(b)$. In short, we are searching for a subset of the bids that will maximize the seller's revenue while allocating each available unit at most once.

Note that the definition of the optimal allocation assumes that bids are additive–that an auction participant who submits multiple bids may be allocated any number of these bids for a price that equals the sum of each allocated bid's price offer. In some cases, however, a participant may wish to submit two or more bids but require that at most one will be allocated. We permit such additional constraints through the use of *dummy goods*, introduced already in (Fujishima, Leyton-Brown, & Shoham 1999). Dummy goods are normal single-unit goods which do not correspond to actual goods in the auction, but serve to enforce mutual exclusion between bids. For example, if bids b_1 and b_2 referring to bundles $e(b_1)$ and $e(b_2)$ are intended to be mutually exclusive, we add a dummy good d to each bid: $e(b_1)$ becomes $e(b_1) \cup d$, and $e(b_2)$ becomes $e(b_2) \cup d$. Since the good d can be allocated only once, at most one of these bids will be in any allocation. (More generally, it is possible to introduce n-unit dummy goods to enforce the condition that no more than n of a set of bids may be allocated.) While dummy goods increase the expressive power of the bidding language, their use has no impact on the optimization algorithm. Hence, in the remainder of this paper we do not discriminate between dummy goods and real goods, and we assume that all bids are additive.

In the sequel, we will also make use of the following notation. Given an allocation π and a good i, we will denote the total number of units allocated in π, and the total number of units of good i allocated in π, by $units(\pi)$ and $units_i(\pi)$ respectively. In addition $units(total)$ will denote the total number of units over all goods.

Algorithm Definition

Branch-and-Bound Search

Given a set of bids, CAMUS systematically compares the revenue from all full allocations in order to determine the optimal allocation. This comparison is implemented as a depth-first search: we build up a partial allocation one bid at a time. Once we have constructed a full allocation we backtrack, removing the most recently added bid from the partial allocation and adding a new bid instead. Sometimes we can safely *prune* the search tree, backtracking before a full allocation has been constructed. Every time a bid is added to the current allocation, CAMUS computes an estimate of the revenue that will be generated by the unallocated goods which remain. Provided that this estimate function $o()$ always provides an upper bound on the actual revenue, we can prune whenever $p(\pi) + o(\pi) \leq p(\pi_{best})$, where π is the current allocation, $p(\pi) = \Sigma_{b\in\pi} p(b)$ and π_{best} is the best allocation observed so far.

Bins

Bins are partitioned sets of bids. Consider some ordering of the goods. There is one bin for each good, and each bid belongs to the bin corresponding to its lowest-order good. During the search we start in the first bin and consider adding each bid in turn. After adding a bid to our partial allocation we move to the bin corresponding to the lowest-order good with any unallocated units. For example, if the first bid we select requests all units of goods 1, 2 and 4, we next proceed to bin 3. Besides making it easy to avoid consideration of conflicting bids, bins are powerful because they allow the pruning function to consider context without significant computational cost. If bids in bin_i are currently being considered then the pruning function must only take into account bids from $bin_i \ldots bin_m$. Because the partitioning of bids into bins does not change during the search we may compute the pruning information for each bin in a preprocessing step.

Subbins

In the multi-unit setting, we will often need to select more than one bid from a given bin. This leads to the idea of *subbins*. A subbin is a subset of the bids in a bin that is constructed during the search. Since subbins are created dynamically they cannot provide precomputed contextual information; rather, they facilitate the efficient selection of multiple bids from a given bin. Every time we add a bid to our partial allocation we create a new subbin containing the next set of bids to consider. If the search moves to a new bin, the new subbin is generated from the new bin by removing all bids that conflict with the current partial allocation. If the search remains in the same bin, the new subbin is created from the current subbin by removing conflicting bids as above, and additionally: if $bid_1, bid_2, \ldots, bid_i$ is the ordered set of elements in the current subbin and bid_j is the bid that was just chosen, then we remove all $bid_k, k \leq j$. In this way we consider all combinations of non-conflicting bids in each bin, rather than all permutations.

Dominated Bids

Some bids may be removed from consideration in a polynomial-time preprocessing step. For each pair of bids (b_1,b_2) where both name the same goods but $p(b_1) \geq p(b_2)$ and $e(b_1)_j \leq e(b_2)_j$ for every good j, we may remove b_2 from the list of bids to be considered during the search, as b_2 is never preferable to b_1 (hence we say that b_1 *dominates* b_2). However, it is possible that an optimal allocation contains both b_1 *and* b_2. For this reason we store b_2 in a secondary data structure associated with b_1, and consider adding it to an allocation only after adding b_1.

Dynamic Programming

Singleton bids (that is, bids that name units from only one good) deserve special attention. These bids will generally be among the most computationally expensive to consider– the number of nodes to search after adding a very short bid is nearly the same as the number of nodes to search after skipping the bid, because a short bid allocates few units and hence conflicts with few other bids. Unfortunately, we expect that singleton bids will be quite common in a variety of real-world multi-unit CA's. CAMUS simplifies the problem of singleton bids by applying a polynomial-time dynamic programming technique as a preprocessing step. We construct a vector $singleton_g$ for each good g, where each element of the vector is a set of singleton bids naming only good g. $singleton_g(j)$ evaluates to the revenue-maximizing set of singleton bids totaling j units of good g. This frees us from having to consider singleton bids individually; instead, we consider only elements of the singleton vector and treat these elements as atomic bids during the search. Also, there is never a need to add more than one element from each singleton vector. To see why, imagine that we add both $singleton_g(j)$ and $singleton_g(k)$ to our partial allocation. These two elements may have bids in common, and additionally there may be singleton bids with more than $max(j,k)$ elements that would not conflict with our partial allocation but that we have not considered. Clearly, we would be better off adding the single element $singleton_g(j+k)$.

Caching

Consider a partial allocation π_1 that is reached during the search phase. If the search proceeds beyond π_1 then $o(\pi_1)$ was not sufficiently small to allow us to backtrack. Later in the search we may reach an allocation π_2 which, by combining different bids, covers exactly the same number of units of the same goods as π_1. CAMUS incorporates a mechanism for caching the results of the search beyond π_1 to generate a better estimate for the revenue given π_2 than is given by $o(\pi_2)$. (Since π_1 and π_2 do not differ in the units of goods that remain, $o(\pi_1) = o(\pi_2)$.) Consider all the allocations extending π_1 upon consideration of which the algorithm backtracked, denoted $s_1, s_2, \ldots, s_f$. When we backtracked at each s_i we did so because $p(s_i)+o(s_i) \leq \pi_{best}$, as explained above. It follows that $max_i(p(s_i)+o(s_i))$ is an overestimate of the revenue attainable beyond π_1, and that it is a smaller overestimate than $o(\pi_1)$ (if it were not, we would have backtracked at π_1 instead). Since in general $p(\pi_1) \neq p(\pi_2)$, we cache the value $max_i(p(s_i)+o(s_i)) - p(\pi_1)$ and backtrack when $p(\pi_2) + cache(\pi_2) \leq p(\pi_{best})$. Our cache is implemented as a hash table, since caching is only beneficial to the overall search if lookup time is inconsequential. A consequence of this choice of data structure is that cache data may sometimes be overwritten; we overwrite an old entry in the cache when the search associated with the new entry examined more nodes. Even when we do overwrite useful data the error is not catastrophic, however: in the worst case we must simply search a subtree that we might otherwise have pruned.

Heuristics

Two ordering heuristics are used to improve CAMUS's performance. First, we must determine an ordering of the goods; that is, which good corresponds to the first bin, which corresponds to the second, etc. For each good i we compute $score_i = \frac{numbids_i \cdot q(i)}{avgunits_i}$, where $numbids_i$ is the number of bids that request good i and $avgunits_i$ is the average number of *total* units (i.e., not just units of good i) requested by these bids. We designate the lowest-order good as the good with the lowest score, then we recalculate the score for the remaining goods and repeat. The intuition behind this heuristic is as follows:

- We want to minimize the number of bids in low-order bins, to minimize early branching and thus to make each individual prune more effective.

- We want to minimize the number of units of goods corresponding to low-order bins, so that we will more quickly move beyond the first few bins. As a result, the pruning function will be able to take into account more contextual information.
- We want to maximize the total number of units requested by bids in low-order bins. Taking these bids moves us more quickly towards the leaves of the search tree, again providing the pruning function with more contextual information.

Our second heuristic determines the ordering of bids within bins. Given current partial allocation π, we sort bids in a given bin in descending order of $score(b_j)$, where $score(b_j) = \frac{p(b_j)}{units(b_j)} + o(\pi \cup b_j)$. The intuition behind this heuristic is that the average price per unit of bid_j is a measure of how promising the bid is, while the pruning overestimate for $o(\pi \cup bid_j)$ is an estimate of how promising the unallocated units are, given the partial allocation. This heuristic helps CAMUS to find good allocations quickly, improving anytime performance and also increasing π_{best}, making pruning more effective. Because the pruning overestimate depends on π, this ordering is performed dynamically rather than as a pre-processing step.

CAMUS Outline

Based on the above, it is now possible to give an outline of the CAMUS algorithm:

- Process dominated bids.
- Determine an ordering on the goods, according to the good-ordering heuristic.
- Using the dynamic programming technique, determine the optimal combination of singleton bids totaling $1 \dots q(j)$ for each good j.
- Partition all non-singleton bids into bins, according to the good ordering.
- Precompute pruning information for each bin.
- Set $i = 1$ and $\pi = \{\}$.
- Recursive entry point:
  - For j = 1 ...number of bids in the current subbin of bin_i.
    - $\pi = \pi \cup bid_j$.
    - If $(p(\pi) + cache(\pi) \leq p(\pi_{best}))$ backtrack.
    - If $(p(\pi) + o(\pi) \leq p(\pi_{best}))$ backtrack.
    - If $(units(\pi) = units(total))$ record π if it is the best; backtrack.
    - Set i to the index of the lowest-order good in π where $units_i(\pi) < q(i)$. (i may or may not change)
    - Construct a new subbin based on the previous subbin of bin_i (which is bin_i itself if i changed above):
      - Include all bid_k from current subbin, where $k > j$.
      - Include all dominated bids associated with bid_j.
      - Include $singleton_i(q(i) - units_i(\pi))$.
      - Sort the subbin according to the subbin-ordering heuristic.
      - Recurse to the recursive entry point, above, and search this new subbin.
    - $\pi = \pi - bid_j$.
  - End For
- Return the optimal allocation: π_{best}.

CAMUS procedures: a closer look

In this section we examine two of CAMUS's fundamental procedures more formally. Additional details will be presented in our full paper.

Pruning

In this subsection we explain the implementation of CAMUS's pruning function and demonstrate that it is guaranteed not to underestimate the revenue attainable given a partial allocation. Consider a point in the search where we have constructed some partial allocation π. The task of our pruning function is to give an upper bound on the optimal revenue attainable from the unallocated items, using the remaining bids (i.e., the bids that may be encountered during the remainder of the search). Hence, in the sequel when we refer to goods, the number of units of a good and bids, we refer to what remains at our point in the search.

First, we provide an intuitive overview. For every (remaining) good j we will calculate a value $v(j)$. Simplifying slightly, this value is the largest average price per unit of all the (remaining) bids requesting units of good j that do not conflict with π, multiplied by the number of (remaining) units of j. The sum of $v(j)$ values for all goods is an upper bound on optimal revenue because it relaxes the constraint that the bids in the optimal allocation may not conflict.

More formally, let $G = \{g_1, g_2, \dots, g_m\}$ be a set of goods. Let $q'(j)$ denote the number of available units of good j. Consider a set of bids $B = \{b_1, \dots, b_n\}$. Bid b_i is associated with a pair $(p(b_i), e(b_i))$ where $p(b_i)$ is the price offer of bid b_i, and $e(b_i) = (e(b_i)_1, e(b_i)_2, \dots, e(b_i)_m)$

where $e(b_i)_j$ is the requested number of units of good j in b_i. For each bid b_i, let $a(b_i) = \frac{p(b_i)}{\Sigma_{1 \leq j \leq m} e(b_i)_j}$ be the average price per unit of bid b_i. Notice that the average price per unit may change dramatically from bid to bid, and it is a non-trivial notion; our technique will work for any arbitrary average price per unit. Let $L(j)$ be a sorted list of the bids that refer to non-zero units of good j; the list is sorted in a monotonically decreasing manner according to the a_i's. Let $|L(j)|$ denote the number of elements in $L(j)$, and let $L(j)_k$ denote the k-th element of $L(j)$.

$v(j)$ is determined by the following algorithm:

Let $v(j)$:=0;
Let $m(j)$:=0;
For $i := 1$ to $|L(j)|$ do
if $m(j) < q'(j)$ then
$\{$let $d := min(e(L(j)_i)_j, q(j) - m(j)); m(j) = m(j) + d; v(j) = v(j) + a(L(j)_i) \cdot d\}$

Theorem 1 *Let $B^o = \{b^0_1, b^0_2, \ldots, b^0_s\}$ be the bids in an optimal allocation. Then, $R^o = \Sigma_{b \in B^o} p(b) \leq \Sigma_{1 \leq j \leq m} v(j)$.*

Sketch of proof: Consider the bid $b^o \in B^o$. Then, $p(b^o) = \Sigma_{1 \leq j \leq m} a(b^o) \cdot e(b^o)_j$. Hence, $R^o = \Sigma_{b \in B^o} p(b) = \Sigma_{b \in B^o} \Sigma_{1 \leq j \leq m} a(b) \cdot e(b)_j$. By changing the order of summation we get that $R^o = \Sigma_{1 \leq j \leq m} \Sigma_{b \in B^o} a(b) \cdot e(b)_j$. Notice that, given a particular j, the contribution of bid b to $\Sigma_{b \in B^o} a(b) \cdot e(b)_j$ is $a(b) \cdot e(b)_j$. Recall now that $v(j)$ has been constructed from the set of all bids that refer to good j by choosing the maximal available units of good j from the bids in $L(j)$, where these bids are sorted according to the average price per unit of good. Hence, we get $v(j) \geq \Sigma_{b \in B^o} a(b) \cdot e(b)_j$. Given that the above holds for every good j, this implies that $\Sigma_{1 \leq j \leq m} v(j) \geq \Sigma_{b \in B^o} p(b)$, as requested.

The above theorem is the central tool for proving the following theorem:

Theorem 2 *CAMUS is complete: it is guaranteed to find the optimal allocation in a multi-unit combinatorial auction problem.*

Pre-Processing of Singletons

In this subsection we explain the construction of the $singleton_g$ vector described above, and demonstrate that $singleton_g(j)$ is the revenue-maximizing set of singleton bids for good g that request a total not exceeding j units.

Let $b_1, b_2, \ldots, b_l$ be bids for a single good g, where the total number of available units of good g is q. Let $p(b_i)$ and $e(b_i)$ be the price offer and the quantity requested by b_i, respectively. Our aim is to compute the optimal selection of b_i's in order to allocate k units of good g, for $1 \leq k \leq q$. Consider a two dimensional grid of size $[1 \ldots l] \times [1 \ldots q]$ where the (i, j)-th entry, denoted by $U(i, j)$, is the optimal allocation of j units considering only bids $b_1, b_2, \ldots, b_i$. The value of $U(i, j)$, denoted by $V(i, j)$, is the sum of the price offers of the bids in $U(i, j)$. $U(1, j)$ will be b_1 if b_1 requests no more than j units, and otherwise will be the empty set. Now we can define $U(i, j)$ recursively:

1. $e(b_i) > j$: $U(i, j) = U(i-1, j)$;
2. $e(b_i) = j$: if $p(b_i) > V(i-1, j)$ then $U(i, j) = b_i$. Else $U(i, j) = U(i-1, j)$.
3. $e(b_i) < j$: if $V(i-1, j) \geq p(b_i) + V(i-1, j-e(b_i))$ then $U(i, j) = U(i-1, j)$. Else $U(i, j) = b_i \cup U(i-1, j-e(b_i))$.

This dynamic programming procedure is polynomial, and yields the desired result; the optimal allocation of k units is given by $U(l, k)$. Set $singleton_g(k) = U(l, k)$, $1 \leq k \leq q$.

Experimental results

Unfortunately, no real-world data exists to describe how bidders will behave in general multi-unit combinatorial auctions, precisely because the determination of winners in such auctions was previously unfeasible. We have therefore tested CAMUS on sets of bids drawn from a random distribution. We created bids as follows, varying the parameters num_{goods} and num_{bids}, and fixing the parameters $units_{max} = 5$, $avgprice_{base} = 50$, $avgprice_{var} = 25$, $prob_1 = 0.8$, $prob_2 = 0.65$, $price_{var} = 0.5$:

1. Set the number of units that exist for each good:
   (a) For each good i, randomly choose $units_i$ from the range $[1 \ldots units_{max}]$.
   (b) If $\Sigma_i units_i \neq \frac{num_{goods} \Sigma_{j=1}^{units_{max}} j}{units_{max}}$ (the expectation on $\Sigma_i units_i$) then go to (a). This ensures that each trial involves the same total number of units.
2. Set an average price for each good: $avgprice_i$ is drawn uniformly randomly from the range $[avgprice_{base} - avgprice_{var} \ldots avgprice_{base} + avgprice_{var}]$.
3. Select the number of goods in the bid. This number is drawn from a decay distribution:
   (a) Randomly choose a good that has not already been added to this bid
   (b) With probability $prob_1$, if more goods remain then go to (a)
4. Select the number of units of each good, according to another decay distribution:
   (a) Add a unit
   (b) With probability $prob_2$, if more units remain then go to (a)
5. Set a price for this bid: $price = rand(1 - price_{var}, 1 + price_{var}) \cdot \Sigma_{i \in bid}(avgprice_i \cdot units_i)$

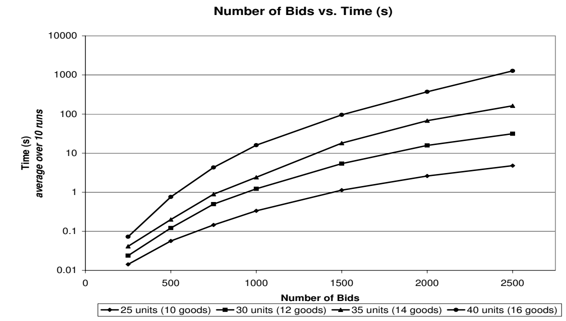
This distribution has the following characteristics that we consider to be reasonable. Bids will tend to request a small number of goods, independent of the total number of goods.

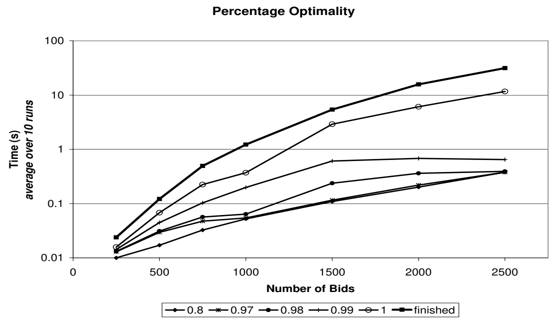
Such data cases are computationally harder than drawing a number of goods uniformly from a range, or than scaling the average number of goods per bid to the maximum number of goods. Likewise, bids will tend to name a small number of units per good. Prices tend to increase linearly in the number of units, for a fixed set of goods. This is a harder case for our pruning technique, much harder than drawing prices uniformly from a range. In fact, it may be reasonable for prices to be superlinear in the number of units, as the motivation for holding a CA in the first place may be that bidders are expected to value bundles more than individual goods. However, this would be an easier case for our pruning algorithm, so we tested on the linear case instead. The construction of realistic, hard data distributions remains a topic for further research.

Our experimental data was collected on a Pentium III-733 running Windows 2000, with 25 MB allocated for CAMUS's cache. Our figure *Number of Bids vs Time* shows CAMUS's performance on the distribution described above, with each line representing runs with a different number of goods. Note that, for example, CAMUS solved problems with 35 objects (14 goods) and 2500 bids in about two minutes, and problems with 25 objects (10 goods) and 1500 bids in about a second. Because the lines in this graph are sublinear on the logarithmic scale, CAMUS's performance is sub-exponential in the number of bids, though it remains exponential in the number of goods. Our figure *Percentage Optimality* shows CAMUS's anytime performance. Each line on the graph shows the time taken to find solutions with revenue that is some percentage of the optimal, calculated after the algorithm terminated. Note that the time taken to *find* the optimal solution is less than the time taken for the algorithm to finish, *proving* that this solution is optimal. These anytime results are very encouraging–note that CAMUS finds a 99% optimal solution an order of magnitude more quickly than it takes for the algorithm to run to completion. This suggests that CAMUS could be useful on much larger problems than we have shown here if an optimal solution were not required.

Conclusions

In this paper we introduced CAMUS, a novel algorithm for determining the optimal set of winning bids in general multi-unit combinatorial auctions. The algorithm has been tested on a variety of data distributions and has been found to solve problems of considerable scale in an efficient manner. CAMUS extends our CASS algorithm for single-unit combinatorial auctions, and enables a wide extension of the class of combinatorial auctions that can be efficiently implemented. In our current research we are studying the addition of random noise into our good and bin ordering heuristics, combined with periodic restarts and the deletion of previously-searched bids, to improve performance on hard cases while still retaining completeness.





References

Fudenberg, D., and Tirole, J. 1991. *Game Theory*. MIT Press.

Fujishima, Y.; Leyton-Brown, K.; and Shoham, Y. 1999. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI-99*.

Lehmann, D.; O'Callaghan, L.; and Shoham, Y. 1999. Truth revalation in rapid, approximately efficient combinatorial auctions. In *ACM Conference on Electronic Commerce*.

Monderer, D., and Tennenholtz, M. 2000. Optimal Auctions Revisited. Artificial Intelligence, forthcoming.

Nisan, N., and Ronen, A. 2000. Computationally feasible vcg mechanisms. To appear.

Nisan, N. 1999. Bidding and allocation in combinatorial auctions. Working paper.

Rothkopf, M.; Pekec, A.; and Harstad, R. 1998. Computationally manageable combinatorial auctions. *Management Science* 44(8):1131–1147.

Sandholm, T. 1999. An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI-99*.

Tennenholtz, M. 2000. Some tractable combinatorial auctions. To appear in the proceedings of AAAI-2000.

Vries, S., and Vohra, R. 2000. Combinatorial auctions: A brief survey. Unpublished manuscript.

Wellman, M.; Wurman, P.; Walsh, W.; and MacKie-Mason, J. 1998. Auction protocols for distributed scheduling. Working paper (to appear in Games and Economic Behavior).