Taming the Computational Complexity of Combinatorial Auctions: Optimal and Approximate Approaches

Yuzo Fujishima, Kevin Leyton-Brown and Yoav Shoham
Computer Science Department, Stanford University, Stanford CA, 94305

fujisima@ccs.mt.nec.co.jp (visiting from NEC Corporation)
kevinlb@cs.stanford.edu
shoham@cs.stanford.edu

Abstract

In combinatorial auctions, multiple goods are sold simultaneously and bidders may bid for arbitrary combinations of goods. Determining the outcome of such an auction is an optimization problem that is NP-complete in the general case. We propose two methods of overcoming this apparent intractability. The first method, which is guaranteed to be optimal, reduces running time by structuring the search space so that a modified depth-first search usually avoids even considering allocations that contain conflicting bids. Caching and pruning are also used to speed searching. Our second method is a heuristic, market-based approach. It sets up a virtual multi-round auction in which a virtual agent represents each original bid bundle and places bids, according to a fixed strategy, for each good in that bundle. We show through experiments on synthetic data that (a) our first method finds optimal allocations quickly and offers good anytime performance, and (b) in many cases our second method, despite lacking guarantees regarding optimality or running time, quickly reaches solutions that are nearly optimal.

1 Combinatorial Auctions

Auction theory has received increasing attention from computer scientists in recent years.[1] One reason is the explosion of internet-based auctions. The use of auctions in business-to-business trades is also increasing rapidly [Cortese and Stepanek, 1998]. Within AI there is growing interest in using auction mechanisms to solve distributed resource allocation problems. For example, auctions and other market mechanisms are used in network bandwidth allocation, distributed configuration design, factory scheduling, and operating system memory allocation [Clearwater, 1996]. Market-oriented programming has been particularly influential [Wellman, 1993; Mullen and Wellman, 1996].

The value of a good to a potential buyer can depend on what other goods s/he wins. We say that there exists complementarity between goods g and h to bidder b if $u_b(\{g,h\}) > u_b(\{g\})+u_b(\{h\})$, where $u_b(G)$ is the utility to b of acquiring the set of goods G. If goods g and h were auctioned separately, it is likely that neither of the typically desired properties for auctions—efficiency and revenue maximization—would hold. One way to accommodate complementarity in auctions is to allow bids for combinations of goods as well as individual goods. Generally, auctions in which multiple goods are auctioned simultaneously and bidders place as many bids as they want for different bundles of goods are called combinatorial auctions[2].

It is also common for bidders to desire a second good less if they have already won a first. We say that there exists substitutability between goods g and h to bidder b when $u_b(\{g,h\}) < u_b(\{g\})+u_b(\{h\})$. A common example of substitutability is for a bidder to be indifferent between several goods but not to want more than one. In order to be useful, a combinatorial auction mechanism should provide some way for bidders to indicate that goods are substitutable.

Combinatorial auctions are applicable to many real-world situations. In an auction for the right to use railroad segments a bidder desires a bundle of segments that connect two particular points; at the same time, there may be alternate paths between these points and the bidder needs only one [Brewer and Plott, 1996]. Similarly, in the FCC spectrum auction bidders may desire licenses for multiple geographical regions at the same frequency band while being indifferent to which particular band they receive [Milgrom, 1998]. The same situation also occurs in military operations when multiple units each have several alternate plans and each plan may require a different bundle of resources.

[1] This material is based upon work supported by DARPA under the CoABS program, contract #F30602-98-C-0214, and by a Stanford Graduate Fellowship.

[2] Auctions in which combinatorial bidding is allowed are alternately called *combinatorial* and *combinational*.

While economics and game theory provide many insights into the potential use of such auctions, they have little to say about computational considerations. In this paper we address the computational complexity of combinatorial auctions.

2 The Complexity Problem

There has been much work in economics and game theory on designing combinatorial auctions. The Clarke-Groves-Vickrey mechanism (also known as the Generalized Vickrey Auction, or GVA) has been particularly influential [Mas-Colell et al., 1995; Varian, 1995]. It is beyond the scope of this paper to review such mechanisms, but they share a central problem: given a collection of bids on bundles, finding a set of non-conflicting bids that maximizes revenue. (A more precise definition is given in Section 3.) This problem is easily shown to be NP-complete[3] [Rothkopf et al., 1995].

Several methods have been conceived to cope with the computational complexity of combinatorial auctions, most aiming to ease the difficulty of finding optimal allocations. They can be classified into three categories based on the strategies they use.

One strategy is to restrict the degree of freedom of bidding to simplify the task of finding optimal allocations. Rothkopf et al. show that an optimal allocation can be found in polynomial time if (1) each bid contains no more than two goods; (2) for any two bids, either they are disjoint or one is a subset of the other; or (3) each bid contains only consecutive goods given a one-dimensional ordering of goods [Rothkopf et al., 1995].

Another strategy is to shift the burden of finding an optimal allocation to bidders. [Banks et al., 1989] and [Bykowsky et al., 1995] have reported a mechanism called AUSM in which non-winning bids are pooled in a stand-by queue. Bidders can combine their bids with other bids currently in the queue to form new allocations. A new allocation is adopted if it generates more revenue than the previously best allocation.

A third strategy is to attempt to find an optimal allocation but to be satisfied with a sub-optimal allocation when the expenditure of further resources becomes unacceptable. In other words, the optimality of the allocation is traded-off with the resources required, especially time.

In this paper we present two algorithms. The first is an anytime algorithm that attempts to exploit a problem's particular bid structure to reduce the size of the search. It also reduces search time by caching partial results and by pruning the search tree. The second algorithm uses a market-based approach to determine an acceptable allocation, although it is not guaranteed to find an optimal one. We then show results of experiments with synthetic data suggesting that these methods, though not provided with formal guarantees, appear to have surprisingly good performance. Additionally, the market-based approach appears to produce allocations that are always optimal or nearly optimal.[4]

[3] The GVA has the additional shortcoming of requiring bidders to submit an unreasonably large number of bids, but we do not address this issue here.

3 Precise Problem Statement

In this paper we propose two methods for finding desirable allocations based on bids submitted. We start by formally defining the optimization problem. Denote the set of goods by G and the set of non-negative real numbers by R^+. A bid $b=(p_b,G_b)$ is an element of $\underline{S}= R^+\times(2^G-\{\emptyset\})$. Let B be a subset of S. A set $F\subseteq B$ is said to be feasible if $\forall b,c\neq b\in F$ $G_b\cap G_c=\emptyset$. Denote the set of all feasible allocations for B by $\Phi(B)$. Further, let $G(B)=\cup_{b\in B}G_b$ be the set of goods contained in the bids of B.

[Problem] Find an allocation $W\in\Phi(B)$ such that $\forall F\in\Phi(B)\ \sum_{b\in F}p_b\leq\sum_{b\in W}p_b$. Such an allocation is said to be optimal or revenue maximizing.

What kind of value interrelation between goods can be represented by the bids defined above? Clearly, complementary values are easily accommodated. Suppose a bidder bids \$20 for each of $\{g\}$ and $\{h\}$, and \$50 for $\{g,h\}$. In this case any revenue-maximizing algorithm will correctly select the $\{g,h\}$ bid instead of $\{g\}$ and $\{h\}$.

This bid format is also sufficient for representing substitutability through an encoding trick. Suppose a bidder is willing to pay \$20 for $\{g\}$ and \$30 for $\{h\}$ but only \$40 for $\{g,h\}$. In this case, bids cannot be submitted as before since the revenue-maximizing algorithm would select the pair $\{g\}$ and $\{h\}$ over $\{g,h\}$, charging the bidder \$50 instead of \$40 for g and h. However, this problem can be solved by the introduction of 'dummy goods'—virtual goods that enforce an exclusive-or relationship. (Each dummy good must appear only in a single bidder's bids.) In our example, the bidder could submit the following bids: (\$20, $\{g,d\}$), (\$30, $\{h,d\}$), and (\$40, $\{g,h\}$) where d is a new, unique dummy good. The first two bids are now mutually exclusive and so will never be allocated together. This technique can lead to a combinatorial explosion in the number of bids if many goods are substitutable, but in many interesting cases this does not arise.

4 CASS Algorithm

When the number of goods and bids is small enough, an exhaustive search can be used to determine the optimal allocation. We propose an algorithm, Combinatorial Auction Structured Search (CASS), presented as a naïve brute-force approach followed by four improvements. CASS considers fewer partial allocations than the brute-force method because it structures the search space to avoid considering allocations containing conflicting bids. It also caches the results of partial searches and prunes the search tree. Finally, it may be used as an any-

[4] We do not analyze the impact of the approximation on the equilibrium strategies in auction mechanisms such as GVA; we will address this issue in a future paper.

time algorithm, as it tends to find good allocations quickly.

4.1 Brute-Force Algorithm

Suppose there are $|G|$ goods 1, 2, ..., $|G|$, and $|B|$ bids 1, 2, …, $|B|$. First, bids that will never be part of an optimal allocation are removed. That is, if for bid $b_k=(p_k,G_k)$ there exists a bid $b_l=(p_l,G_l)$ such that $p_l > p_k$ and $G_l \subseteq G_k$, then b_k is removed because it can always be replaced by b_l, increasing revenue. Then for each good g, if there is no bid $b=(x,\{g\})$ a dummy bid $b=(0,\{g\})$ is added.

Our brute-force algorithm examines all feasible allocations through a depth-first search. Let x be the first bid and y be the last bid. Our implementation follows:

1. If x does not conflict with the current allocation, add x to the current allocation
2. Increment x
3. If more bids can be added to the allocation, go to 2.
4. Update best revenue and allocation observed so far.
5. If y is contained in the current allocation, remove it, set x=y+1 and repeat from 2.
6. Decrement y.
7. If y is not the first bid, go to 5.

4.2 Improvement #1: Bins

A great deal of unnecessary computation is avoided in the brute-force algorithm by checking whether bids conflict with the current allocation before they are added. However, work is still required to determine that a combination is infeasible and to move on to the next bid. It would be desirable to structure the search space to reduce the number of infeasible allocations that are considered in the first place.

We can reduce the number of infeasible allocations considered by sorting bids into bins, D_i, containing all bids b where good $i \in G_b$ and for all j such that $j \in [1, i\text{-}1]$, $j \notin G_b$. Rather than always trying to add each bid to our allocation, we add at most one bid from every bin since all bids in a given bin are mutually exclusive.

In fact, we can often skip bins entirely. While considering bin D_i, if we observe that good $j>i$ is already part of the allocation then we do not need to consider any of the bids in D_j. In general, instead of considering each bin in turn, skip to D_k where $k \notin G(F)$ and $\forall i<k$, $i \in G(F)$.

4.3 Improvement #2: Caching

Let F_i be the partial allocation under consideration when D_i is reached during a search. Define $C_i \subseteq G(F_i)$ where $\forall j \in G(F_i)$, $j>i \leftrightarrow j \in C_i$. Note that there are many different partial allocations F_{i1}, F_{i2}, etc., that share the same C_i, and that if $C_{i1}=C_{i2}$ then the search trees for F_{i1} and F_{i2} are identical beyond D_i. It is therefore possible to cache partial searches based on C_i. However, caching all possible values of C_i would require a cache of size $2^{|G|-(i-1)}$, which would quickly become infeasible. Therefore, we only cache when C_i includes no more than k goods, where k is a threshold defined at runtime for each bin. D_i requires a cache of size $\sum_{j=0}^{k} \binom{|G|-i}{j}$.

4.4 Improvement #3: Pruning #1

Performance can be improved by backtracking whenever a given search path is provably unable to lead to a new best allocation. We can prune whenever $C(F_{i1}) \subset C(F_{i2})$ and $p(F_{i2}) + p(cache(F_{i1})) \leq bestAllocation$. In this case, the sum of the revenue from the cached path beyond F_{i1} and the revenue leading up to F_{i2} is less than the revenue from the best allocation seen so far. Since F_{i1} allocates a superset of the goods allocated in F_{i2} (thus overestimating revenue), a better allocation would not be found by expanding F_{i2}.

4.5 Improvement #4: Pruning #2

We can also backtrack when it is provably impossible to add any bids to the current allocation to generate more revenue than the current best allocation. Before starting the search we calculate an overestimate of the revenue that can be achieved with each good, $o(g) = \max_{b|g \in b} p(b)/|G_b|$. $o(g)$ is the largest average price per bid of bids containing good g. We backtrack at any point during the search with allocation F if $p(F) + \sum_{g \notin F} o(g) \leq p(\text{best_allocation})$. This technique is most effective when good allocations are found quickly. Finding good allocations quickly is also useful if a solution is required before the algorithm has completed (i.e., if CASS is used as an anytime algorithm). We have found that good allocations are found early in the search when the bids in each bin are ordered in descending order of average price per good. Similarly, the pruning technique is most effective when the unallocated goods are those with the lowest $o(g)$ values. To achieve this, we reorder bins so that for any two bins i and j, $o(g_i) > o(g_j) \leftrightarrow i < j$.

5 VSA Algorithm

Our second algorithm is called Virtual Simultaneous Auction (VSA). This market-based method was inspired by market-oriented programming [Wellman, 1993; Mullen and Wellman, 1996] and the simultaneous ascending auction [Milgrom, 1998]. VSA generates a virtual simultaneous auction from the bids submitted in a real combinatorial auction, then simulates the virtual auction to find a good allocation of goods in the real auction.

5.1 Algorithm

First, a virtual simultaneous auction is generated based on the bids submitted in a real combinatorial auction. For each bid $b=(p_b,G_b)$ a virtual bidder v_b is created. The virtual bidders compete in a virtual simultaneous auction that has multiple rounds. Each virtual bidder v_b tries to win all the goods in G_b for the price p_b on an all-or-nothing basis. The virtual auction starts with no goods allocated and the

prices of all goods set to zero. The simultaneous auction is repeated round by round until either an optimal allocation is found or a pre-set time deadline is reached. In the latter case the current best allocation is adopted as the final result.

Each round of VSA has three phases: the virtual auction phase, the refinement phase and the update phase. In the virtual auction phase each virtual bidder bids for the goods they want. Each individual good is allocated to the highest bidder. If a bidder succeeds in winning all desired goods, that bidder becomes a temporary winner. Otherwise the bidder becomes a temporary loser and returns all allocated goods to the auctioneer. In the refinement phase each of the losers is examined in a random order to see whether making that agent a temporary winner (and consequently making a different winner into a loser) would increase global revenue. If so, the list of winners is updated. Finally in the update phase the current highest price of each good is changed to reflect the price that its current winner bid. The current highest price for unallocated goods is reset to zero.

Virtual bidders in VSA follow a simple strategy. If a bidder was the temporary winner in the previous round, the bidder does not bid in the current round. Otherwise, agents calculate the sum of the current highest prices of the goods required. If the sum exceeds an agent's budget, the agent does not bid because the agent will not be able to acquire all the goods simultaneously. If the sum is less than the budget, the agent bids such that the surplus (budget - sum) is equally divided among the goods.

5.2 Properties

In certain circumstances, VSA will find an optimal allocation. Additionally, it is sometimes possible to detect if an optimal allocation has been found, allowing the virtual auction to end before the deadline.

[Theorem] If no virtual bidder bids in a round in the virtual auction, the current set of winners is optimal.

[Proof] Assume that no agents bid in a given round. Define the function that calculates the revenue of an allocation F by $r(F)=\sum_{b\in F}p_b$ and let O denote the optimal set of winners. Split the current set of winners W into two parts O_1 and W_2 such that $O_1=O\cap W$ and $W_2=W\cap\neg O_1$. Also split O into O_1 and O_2 such that O_1 is defined as before and $O_2 = O \cap \neg O_1$. Further, split G into G_1 and G_2 such that $G_1=\cup_{b\in O1}G_b$ and $G_2=G\cap\neg G_1$. By the assumption, for each currently losing bidder, the sum of the current highest prices of the goods needed exceeds the bidder's budget. This is especially true for bidders in O_2, i.e., $\forall b\in O_2$ $p_b<\sum_{g\in Gb}h_g$ where h_g is the current highest price of good g. It follows that $r(O_2) = \sum_{b\in O2}p_b \le \sum_{b\in O2}\sum_{g\in Gb}h_g \le \sum_{g\in G2}h_g = \sum_{b\in W2}\sum_{g\in Gb}h_g = \sum_{b\in W2}p_b = r(W_2)$. (Remember that the minimum price of a good that is not allocated to any agent is zero and agents always bid their entire budgets.) The inequality means that W is optimal because $r(O) = r(O_1)+r(O_2) \le r(O_1)+r(W_2) = r(W)$.

However, there is no guarantee that auctions will always finish, even if an optimal allocation is found.

[Theorem] There exists a set of bids B such that at least one virtual bidder always bids in every round of the virtual auction no matter what bidding strategy is used.

[Proof] Suppose $B=\{a,b,c\}$ where $a=\{p_a, \{1,2\}\}$, $b=\{p_b, \{2,3\}\}$, and $c=\{p_c, \{3, 1\}\}$. Suppose further that $p_a < p_b + p_c$, $p_b < p_c + p_a$, and $p_c < p_a + p_b$. Because the real bids are mutually exclusive, at most one virtual bidder becomes the temporary winner. If none is winning, $h_1=h_2=h_3=0$ and all the bidders bid in the current round. Assume here that bidder a is currently winning. Then $h_1+h_2=p_a$ and $h_3=0$. Assume that neither b nor c bids in the current round. Then for each of b and c, the sum of the prices of goods needed must be larger than or equal to the budget, i.e., $h_2+h_3=h_2\ge p_b$ and $h_3+h_1=h_1\ge p_c$. This means that $p_a = h_1+h_2\ge p_b+p_c$ and contradicts $p_a<p_b+p_c$. This argument doesn't depend on the bidding strategy as long as an agent bids if and only if their budget exceeds the sum of the minimum prices of the goods needed.

It is this property that makes the refinement phase of VSA important. Consider the case $B=B_1\cup B_2\cup...$ where $\forall i,j\ G(B_i)\cap G(B_j)=\emptyset$, $|B_i|=3$ and each B_i satisfies the condition from the proof above. If we omit the refinement phase then the winner in each subset changes every round except the case where there is no winner. Therefore, an optimal global allocation is examined only when in every subset the optimal winner is temporarily winning. Such synchronization is unlikely to occur unless the number of subsets is very small. The refinement phase causes the optimal winners to become the temporary winners in every round, leading to an optimal allocation even though it is not detected as optimal. (In some cases where $\exists i,j$ $G(B_i)\cap G(B_j)\neq\emptyset$ or $|B_i| > 3$ an optimal allocation may be impossible to achieve regardless of the time limit.)

6 Experimental Evaluation

As we have not yet determined each algorithm's formal complexity characteristics we conducted empirical tests. We evaluated (1) how running time varies with the number of bids, and (2) how percentage optimality of the best allocation varies with time, given a particular bid distribution and a fixed number of bids and goods.

6.1 Assumptions and Parameters

The space of this problem is large. Roughly speaking it has three degrees of freedom: the number of goods, the number of bids and the distribution of bids. Most problematic among these is the distribution. Precisely because of the computational complexity of combinatorial auctions there is little or no real data available. In the absence of such data we tested our algorithms against bids drawn randomly from specific distributions.

Throughout the experiments we used the following two distribution functions to determine how often a bid for n goods appears. The first is binomial, $f_b(n)=p^n(1-p)^{N-n}N!/(n!(N-n)!)$, $p=0.2$, in which the prob-

ability of each good being included in a given bid is independent of which other goods are included. The second distribution is of exponential form, $f_e(n)=Ce^{-x/p}$, $p=5$, representing the case where a bid for $n+1$ goods appears $e^{-1/p}$ times less often than a bid for n goods. The prices of bids for n goods is uniformly distributed between $[n(1-d), n(1+d)]$, $d=0.5$.

We do not present any experiments varying the number of goods in this paper because of space constraints. We found that for both CASS and VSA running time increased exponentially with the number of goods.

We ran our experiments on a 450MHz Pentium II with 256MB of RAM, running Windows NT 4.0. 30 MB of RAM was used for the CASS cache. All algorithms were implemented in C++.

6.2 Results

To answer question (1) we measured the running time of CASS, VSA and the brute-force algorithm. Since VSA is not guaranteed to reach the optimal revenue, it was passed this value—calculated by CASS—and stopped when it found an allocation with revenue of at least 95% of optimal. All the results reported here are averages over 10 different runs. Figure 1 shows running time as a function of the number of bids with a binomial distribution, with the number of goods fixed at 30. Figure 2 shows the same thing for an exponential distribution, without the brute-force algorithm. To answer question (2), we measured the optimality of the output of both VSA and CASS as a function of time. Figure 3 shows both algorithms' performance with 15000 bids for 150 goods with a binomial distribution and Figure 4 shows 4500 bids for 45 goods with an exponential distribution.

6.3 Discussion

CASS demonstrates excellent performance both in finding optimal allocations and as an anytime algorithm. In Figures 1 and 2 CASS remains roughly an order of magnitude faster than VSA as the number of bids increases. Both curves appear to grow sub-linearly on the logarithmic graph, suggesting polynomial-time performance. As the size of the problem is increased (Figures 3 and 4) CASS still performs better than VSA for the binomial distribution, but initially offers worse anytime performance for the exponential distribution. These results—and other experiments we have conducted—suggest that VSA is most likely to outperform CASS when the number of goods is relatively large compared to average bid length. (Note that VSA runs to a time limit, so the point at which VSA's curve ends is not meaningful.)

CASS's effectiveness is strongly influenced by the distribution of bids, particularly as the number of goods increases. If bids contain a large number of goods on average, improvement #1 will have a substantial effect because more bins will be skipped between every pair of bins that are considered, eliminating the need to individually examine all the bids in those bins. However, our

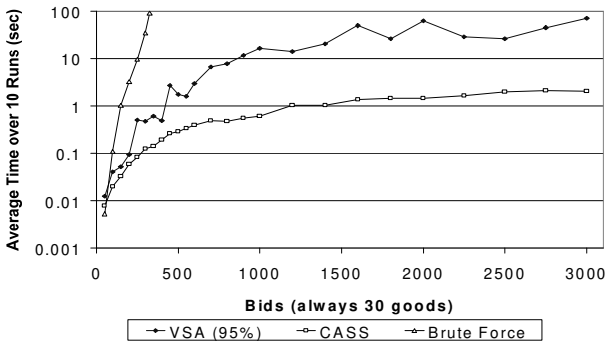

Figure 1: Running Time Comparison (Binom. Dist.)

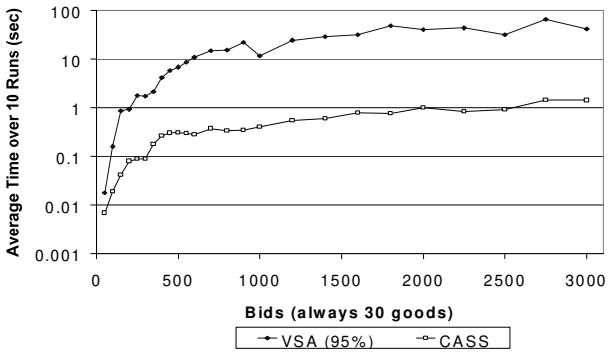

Figure 2: Running Time Comparison (Exp. Dist.)

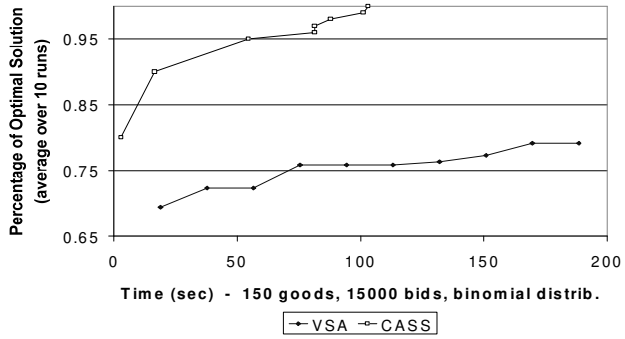

Figure 3: Anytime Behavior (Binom. Dist.)

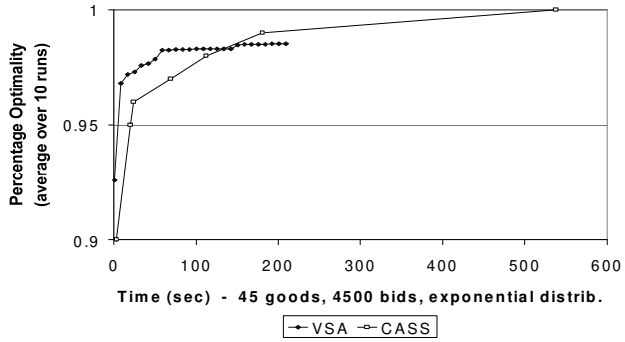

Figure 4: Anytime Behavior (Exp. Dist.)

caching scheme favors distributions with small bids because they increase the likelihood that partial allocations will be cacheable. The pruning technique described in 4.4 reduces the number of nodes that are cached, lowering

memory consumption and making CASS feasible for larger problems. Our second pruning technique often improves performance by two orders of magnitude, though it is most effective when the variance of average price per bid is relatively small. This technique also reduces the optimal cache size, further reducing memory consumption. As a result of pruning, with pruning the amount of memory available for caching does not seem to be a limiting factor in CASS's performance.

VSA is interesting for two reasons. Firstly, it appears to offer good anytime performance in cases with small bids and many goods. Secondly, it provides a case study in the power of market-based optimization. Further work is needed to reach firm conclusions, but it appears that as a centralized optimization method VSA is overshadowed by other techniques. However, other attractions of market-based optimization—in particular its inherent distributed nature and robustness to change in problem specification—may make VSA attractive for some domains.

7 Related and Ongoing Work

As far as we are aware, the work most directly relevant to the ideas presented here is a paper by Sandholm [1999] that appears in these proceedings. Sandholm's Bidtree algorithm appears to be closely related to CASS, but important differences hold. In particular, Bidtree performs a secondary depth-first search to identify non-conflicting bids, whereas CASS's structured approach allows it to avoid considering most conflicting bids. Bidtree also performs no pruning analogous to our Improvement #3 and no caching. On the other hand, Bidtree uses an IDA* search strategy rather than CASS's branch-and-bound approach, and does more preprocessing. We intend to continue studying the differences between these algorithms, including differences in experimental settings.

Our problem can of course be abstracted away from the auction motivation and viewed as a straightforward combinatorial optimization. This suggests a wealth of literature that could be applied. We are currently implementing some of these techniques and comparing them to our present results. We are especially interested in comparisons with mixed-integer programming and greedy methods. In particular, we have been investigating a new algorithm[5] that orders bids in descending order according to average price per good, and does a depth-first search with extensive pruning. This algorithm appears to offer performance similar to CASS, and we intend to report on it in a follow-up paper.

8 Conclusion

We have proposed two novel algorithms to mitigate the computational complexity of combinatorial auctions.

CASS determines optimal allocations very quickly, and also provides good anytime performance. In the future we intend to pursue a formal analysis of CASS's computational complexity, and to test both CASS and VSA with data collected from real bidders.

VSA can determine near-optimal allocations even in cases with hundreds of goods and tens of thousands of bids. Since it has been infeasible to run CASS on much larger problems we do not yet know how close VSA comes to optimality in these cases. An investigation of VSA's limits remains an area for future work.

[5] This ongoing work is joined by Liadan O'Callaghan and Daniel Lehmann.

References

[Banks, et al., 1989] Jeffrey S. Banks, John O. Ledyard, and David P. Porter. Allocating uncertain and unresponsive resources: an experimental approach. *RAND Journal of Economics*, 20(1): 1-25, 1989.

[Brewer and Plott, 1996] P.J. Brewer and C.R. Plott. A binary conflict ascending price (BICAP) mechanism for the decentralized allocation of the right to use railroad tracks. *International Journal of Industrial Organization*, 14:857-886, 1996]

[Bykowsky et al., 1995] Mark M. Bykowsky, Robert J. Cull, and John O. Ledyard. Mutually destructive bidding: the FCC auction design problem. Social science working paper 916, California Institute of Technology, 1995.

[Clearwater, 1996] Scott H. Clearwater, editor. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[Cortese and Stepanek, 1998] Amy E. Cortese and Marcia Stepanek. Good-bye to fixed pricing? *Business Week*, pages 71-84, May 4, 1998.

[Mas-Colell et al., 1995] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.

[Milgrom, 1998] Paul Milgrom. Putting auction theory to work: the simultaneous ascending auction. Working paper 98-002, Dept. Economics, Stanford University, 1998.

[Mullen and Wellman, 1996] Tracey Mullen and Michael P. Wellman. Some issues in the design of the market-oriented agents. In M.J. Wooldridge, J.P. Muller, and M. Tambe, editors, *Intelligent Agents II: Agent Theories, Architectures, and Languages*. Springer-Verlag, 1996.

[Rothkopf et al., 1995] Michael H. Rothkopf, Aleksandar Pekeč, and Ronald M. Harstad. Computationally manageable combinatorial auctions. DIMACS Technical Report 95-09, April 1995.

[Sandholm, 1999] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, 1999.

[Varian, 1995] Hal R. Varian. Economic mechanism design for computerized agents. In *Proceedings of the First Usenix Conference on Electronic Commerce*, New York, July 1995.

[Wellman, 1993] Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1-23, 1993.