CABOB: A Fast Optimal Algorithm for Combinatorial Auctions[*]

Tuomas Sandholm
sandholm@cs.cmu.edu
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Subhash Suri
suri@cs.ucsb.edu
Department of Computer Science
University of California
Santa Barbara, CA 93106

Andrew Gilpin **David Levine**
{agilpin,dlevine}@CombineNet.com
CombineNet, Inc.
311 S. Craig St
Pittsburgh, PA 15213

Abstract

Combinatorial auctions where bidders can bid on bundles of items can lead to more economical allocations, but determining the winners is $\mathcal{NP}$-complete and inapproximable. We present CABOB, a sophisticated search algorithm for the problem. It uses decomposition techniques, upper and lower bounding (also across components), elaborate and dynamically chosen bid ordering heuristics, and a host of structural observations. Experiments against CPLEX 7.0 show that CABOB is usually faster, never drastically slower, and in many cases drastically faster. We also uncover interesting aspects of the problem itself. First, the problems with short bids that were hard for the first-generation of specialized algorithms are easy. Second, almost all of the CATS distributions are easy, and become easier with more bids. Third, we test a number of random restart strategies, and show that they do not help on this problem because the run-time distribution does not have a heavy tail (at least not for CABOB).

1 Introduction

Auctions are important mechanisms for resource and task allocation in multiagent systems. In many auctions, a bidder's valuation for a combination of distinguishable items is not the sum of the individual items' valuations—it can be more or less. *Combinatorial auctions (CAs)* where bidders can bid on bundles of items allow bidders to express *complementarity* (and, with a rich enough bidding language, also *substitutability* among the items [Sandholm, 1999; Fujishima *et al.*, 1999; Sandholm, 2000; Nisan, 2000]). This expressiveness can lead to more economical allocations of the items because bidders do not get stuck with partial bundles of low value. This has been demonstrated, for example, in airport landing slot allocation [Rassenti *et al.*, 1982], and in transportation exchanges [Sandholm, 1993].

However, determining the winners in a combinatorial auctions is computationally complex. There has recently been a surge of research into addressing that [Rothkopf *et al.*, 1998; Sandholm, 1999; Fujishima *et al.*, 1999; Lehmann *et al.*, 1999; Sandholm and Suri, 2000; Andersson *et al.*, 2000; Hoos and Boutilier, 2000; de Vries and Vohra, 2000]. In this paper we present a fast optimal search algorithm for the problem. Section 2 defines the problem. Our algorithm is presented in Section 3. Section 4 discusses bid ordering heuristics. Experimental results are presented in Sections 5–7. Random restart strategies are discussed in Section 8. Section 9 presents conclusions and future research directions.

[*]This work was funded by, and conducted at, CombineNet, Inc.

2 The Winner Determination Problem

In this section we define the winner determination problem.

Definition 1 *The auctioneer has a set of items,* $M = \{1, 2, \ldots, m\}$, *to sell, and the buyers submit a set of bids,* $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$. *A bid is a tuple* $B_j = \langle S_j, p_j \rangle$, *where* $S_j \subseteq M$ *is a set of items and* $p_j \geq 0$ *is a price. The* binary combinatorial auction winner determination problem *is to label the bids as winning or losing so as to maximize the auctioneer's revenue under the constraint that each item can be allocated to at most one bidder:*

$$\max \sum_{j=1}^{n} p_j x_j \quad \textit{s.t.} \quad \sum_{j|i \in S_j} x_j \leq 1,\ i \in \{1..m\}$$
$$x_j \in \{0, 1\}$$

This is $\mathcal{NP}$-complete [Rothkopf *et al.*, 1998], and it cannot even be approximated to a ratio of $n^{1-\epsilon}$ in polynomial time (unless $\mathcal{P} = \mathcal{NP}$) [Sandholm, 1999].

If bids could be accepted partially, the problem would become a linear program (LP) which can be solved in polynomial time. Here we present the LP-formulation and its dual because we will use them in several ways in our algorithm.

$$\begin{array}{ll} LP & DUAL \\ \max \displaystyle\sum_{j=1}^{n} p_j x_j & \min \displaystyle\sum_{i=1}^{m} y_i \\ \displaystyle\sum_{j|i \in S_j} x_j \leq 1,\ i \in \{1..m\} & \displaystyle\sum_{i \in S_j} y_i \geq p_j,\ j \in \{1..n\} \\ x_j \in \Re & y_i \in \Re \end{array}$$

In this continuous setting, the *shadow price* y_i gives the price for each individual item i.[1] In the binary case individual items cannot generally be given prices, but each y_i value from DUAL gives an upper bound on the price of item i.

[1]If there are some items that are not included in any bids, we have to add to the DUAL the constraint $y_i \geq 0$ for those items. Alternatively (and preferably), such items can simply be removed as a preprocessing step.

3 Description of the Algorithm

Our algorithm, *CABOB (Combinatorial Auction Branch On Bids)*, is a tree search algorithm that branches on bids. The high-level idea of branching on bids was already proposed by [Sandholm and Suri, 2000] in the BOB algorithm. However, BOB was not implemented. CABOB incorporates many of the techniques proposed in BOB and a host of additional ones. All of them have been implemented.

The skeleton of CABOB is a depth-first branch-and-bound tree search that branches on bids. The value of the best solution found so far is a global variable $\tilde{f}^*$. Initially, $\tilde{f}^* = 0$.

A data structure called the *bid graph* is maintained. We denote it by G. The nodes of the graph correspond to bids that are still available to be appended to the search path, i.e., bids that do not include any items that have already been allocated. Two vertices in G share an edge whenever the corresponding bids share items.[2] As vertices are removed from G when going down a search path, the edges that they are connected to are also removed. As vertices are re-inserted into G when backtracking, the edges are also reinserted.

The following pseudocode of CABOB makes calls to several special cases which will be introduced later. For readability, we omit how the solution (set of winning bids) is updated in conjunction with every update of $\tilde{f}^*$.

As will be discussed later, we use a technique for pruning across independent subproblems (components of G). To support this, we use a parameter, MIN, to denote the minimum revenue that the call to CABOB must return (not including the revenue from the path so far or from neighbor components) to be competitive with the best solution found so far. The revenue from the bids that are winning on the search path so far is called g. It includes the lower bounds (or actual values) of neighbor components of each search node on the path so far.

The search is invoked by calling $CABOB(G, 0, 0)$.

Algorithm 3.1 $CABOB(G, g, MIN)$

```
1. Apply special cases COMPLETE and NO_EDGES
2. Run DFS on G; let $c$ be number of components found,
   and let $G_1, G_2, ..., G_c$ be the $c$ independent bid graphs
3. Calculate an upper bound $U_i$ for each component $i$
4. If $\sum_{i=1}^{c} U_i \leq MIN$, then return $0$
5. Apply special case INTEGER
6. Calculate a lower bound $L_i$ for each component $i$
7. $\Delta \leftarrow g + \sum_{i=1}^{c} L_i - \tilde{f}^*$
8. If $\Delta > 0$, then
       $\tilde{f}^* \leftarrow \tilde{f}^* + \Delta$
       $MIN \leftarrow MIN + \Delta$
9. If $c > 1$ then goto (11)
10. Choose next bid $B_k$ to branch on (use articulating bids
    first if any)
       10.a. $G \leftarrow G - \{B_k\}$
       10.b. For all $B_j$ s.t. $B_j \neq B_k$ and $S_j \cap S_k \neq \emptyset$,
             $G \leftarrow G - \{B_j\}$
       10.c. $\tilde{f}^*_{old} \leftarrow \tilde{f}^*$
       10.d. $f_{in} \leftarrow CABOB(G, g + p_k, MIN - p_k)$
       10.e. $MIN \leftarrow MIN + (\tilde{f}^* - \tilde{f}^*_{old})$
       10.f. For all $B_j$ s.t. $B_j \neq B_k$ and $S_j \cap S_k \neq \emptyset$,
             $G \leftarrow G \cup \{B_j\}$
       10.g. $\tilde{f}^*_{old} \leftarrow \tilde{f}^*$
       10.h. $f_{out} \leftarrow CABOB(G, g, MIN)$
       10.i. $MIN \leftarrow MIN + (\tilde{f}^* - \tilde{f}^*_{old})$
       10.j. $G \leftarrow G \cup \{B_k\}$
       10.k. Return $\max\{f_{in}, f_{out}\}$
11. $F^*_{solved} \leftarrow 0$
12. $H_{unsolved} \leftarrow \sum_{i=1}^{c} U_i$,   $L_{unsolved} \leftarrow \sum_{i=1}^{c} L_i$
13. For each component $i \in \{1, \ldots, c\}$ do
       13.a. If $F^*_{solved} + H_{unsolved} \leq MIN$, return $0$
       13.b. $g'_i \leftarrow F^*_{solved} + (L_{unsolved} - L_i)$
       13.c. $\tilde{f}^*_{old} \leftarrow \tilde{f}^*$
       13.d. $f^*_i \leftarrow CABOB(G_i, g + g'_i, MIN - g'_i)$
       13.e. $MIN \leftarrow MIN + (\tilde{f}^* - \tilde{f}^*_{old})$
       13.f. $F^*_{solved} \leftarrow F^*_{solved} + f^*_i$
       13.g. $H_{unsolved} \leftarrow H_{unsolved} - U_i$
       13.h. $L_{unsolved} \leftarrow L_{unsolved} - L_i$
14. Return $F^*_{solved}$
```

We now discuss the techniques of CABOB in more length.

Upper Bounding

In step (3), CABOB uses an upper bound on the revenue the unallocated items can contribute. If the current solution cannot be extended to a new optimal solution under the optimistic assumption that the upper bound is met, CABOB prunes the search path.

Any technique for devising an upper bound could be used here. We solve the remaining LP, whose objective function value gives an upper bound. CABOB does not make copies of the LP table, but incrementally adds (deletes) rows from the LP table as bids are removed (re-inserted) into G as the search proceeds down a path (backtracks). Also, as CABOB moves down a search path, it remembers the LP solution from the parent and uses it as a starting solution for the child's LP.

It is not always necessary to run the LP to optimality. Before starting the LP, one could look at the condition in step (4) to determine the minimum revenue the LP has to produce so that the search branch would not be pruned.[3] Once the LP solver finds a solution that exceeds the threshold, it could be stopped without pruning the search branch. If the LP solver does not find a solution that exceeds the threshold and runs to completion, the branch could be pruned. However, CABOB always runs the LP to completion since it uses the solutions from the LP and the DUAL in several ways.

[2]Since G can be constructed incrementally as bids are submitted, its construction does not add to winner determination time after the auction closes. Therefore, in the experiments, the time to construct G is not included (in almost all cases it was negligible anyway, but for instances with very long bids it sometimes took almost as much time as the search).

[3]In the case of multiple components, when determining how high a revenue one component's LP has to return, the exact solution values from solved neighbor components would be included, as well as the upper bounds from the unsolved neighbor components.

Our experiments showed that the upper bound from LP is significantly tighter than those proposed for previous combinatorial auction winner determination algorithms [Sandholm, 1999; Fujishima *et al.*, 1999; Sandholm and Suri, 2000]. The time taken to solve the LP at every node is negligible compared to the savings in search due to enhanced pruning.

The INTEGER special case

If the LP happens to return integer values ($x_j = 0$ or $x_j = 1$) for each bid j (this occurs more often than we expected), CABOB makes the bids with $x_j = 1$ winning, and those with $x_j = 0$ losing. This is clearly an optimal solution for the remaining bids. CABOB updates $\tilde{f}^*$ if the solution is better than the best so far. CABOB then returns from the call without searching further under that node.

It is easy to show that if some of the x_j values are not integer, we cannot simply accept the bids with $x_j = 1$. Neither can we simply reject the bids with $x_j = 0$. Either approach can compromise optimality.

Lower Bounding

In step (6), CABOB calculates a lower bound on the revenue that the remaining items can contribute. If the lower bound is high, it can allow $\tilde{f}^*$ to be updated, leading to more pruning and less search in the subtree rooted at that node.

Any lower bounding technique could be used here. We use the following rounding technique. In step (3), CABOB solves the remaining LP anyway, which gives an "acceptance level" $x_j \in [0, 1]$ for every remaining bid B_j. We insert all bids with $x_j > 0.5$ into the lower bound solution. We then try to insert the rest of the bids in decreasing order of x_j, skipping bids that share items with bids already in the lower bound. It is easy to prove that this method gives a lower bound.

While rounding techniques are known to provide reasonably good lower bounds on average, in the future we are planning to try other lower bounding techniques within CABOB such as stochastic local search [Hoos and Boutilier, 2000].

Exploiting decomposition

In step (2), CABOB runs an $O(|E| + |V|)$ time depth-first-search (DFS) in the bid graph G. Each tree in the depth-first forest is a connected component of G. Winner determination is then conducted in each component independently. Since search time is superlinear in the size of G, this decomposition leads to a time savings. The winners are determined by calling CABOB on each component separately. As the experiments show, this can lead to a drastic speedup.

Upper and lower bounding across components

In addition to regular upper and lower bounding, somewhat unintuitively, we can achieve further pruning, without compromising optimality, by exploiting information across the independent components. When starting to solve a component, CABOB checks how much that component would have to contribute to revenue in the context of what is already known about bids on the search path so far *and the neighboring components*. Specifically, when determining the MIN value for calling CABOB on a component, the revenue that the current call to CABOB has to produce (the current MIN value), is decremented by the revenues from solved neighbor components and the lower bounds from unsolved neighbor components. *Our use of a MIN value allows the algorithm to work correctly even if on a single search path there may be several search nodes where decomposition occurred, interleaved with search nodes where decomposition did not occur.*

Every time a better global solution is found and $\tilde{f}^*$ is updated, all MIN values in the search tree should be incremented by the amount of the improvement since now the bar of when search is useful has been raised. CABOB handles these updates without separately traversing the tree when an update occurs. CABOB directly updates MIN in step (8), and updates the MIN value of any parent node after the recursive call to CABOB returns.

CABOB also uses lower bounding across components. At any search node, the lower bound includes the revenues from the bids that are winning on the path, the revenues from the solved neighbor components of search nodes on the path, the lower bounds of the unsolved neighbor components of search nodes on the path, and the lower bound on the revenue that the unallocated items in the current search node can contribute.

Due to upper and lower bounding across components (and due to updating of $\tilde{f}^*$), the order of tackling the components can potentially make a difference in speed. CABOB currently tackles components in the order that they are found in the DFS. We plan to study more elaborate component ordering in future research.

Forcing a decomposition via articulation bids

In addition to checking whether a decomposition has occurred, CABOB strives for a decomposition. In the bid choice in step (10), it picks a bid that leads to a decomposition, if such a bid exists. Such bids whose deletion disconnects G are called *articulation bids*. Articulation bids are identified in $O(|E| + |V|)$ time by a slightly modified DFS in G, as proposed in [Sandholm and Suri, 2000].

The scheme of always branching on an articulation bid, if one exists, is often at odds with price-based bid ordering schemes, discussed later. As proved in [Sandholm and Suri, 2000], no scheme from the articulation-based family dominates any scheme from the price-based family, or vice versa, in general. However, our experiments showed that in practice it almost always pays off to branch on articulation bids if they exist (because decomposition reduces search drastically).

Even if a bid is not an articulation bid, and would not lead to a decomposition if the bid is assigned losing, it might lead to a decomposition if it is assigned winning because that removes the bid's neighbors from G as well. This is yet another reason to assign a bid that we branch on to be winning before assigning it to be losing (value ordering). Also, in bid ordering (variable ordering), one could give first preference to articulation bids, second preference to bids that articulate on the winning branch only, and third preference to bids that do not articulate on either branch (among them, price-based bid ordering could be used). One could also try to identify *sets* of bids that articulate the bid graph, and branch on all of the bids in the set. However, to keep the computation linear time in the size of G, CABOB simply gives first priority to articulation bids, and if there are none, uses other bid ordering schemes, discussed later. If there are several articulation bids, CABOB branches on the one that is found first (the others will be found at subsequent levels of the search). One could also use a more elaborate scheme for choosing among articulation bids.

The COMPLETE special case

In step (1), CABOB checks whether the bid graph G is complete: $|E| = |\frac{n(n-1)}{2}|$. If so, only one of the remaining bids can be accepted. CABOB thus picks the bid with highest price, updates $\tilde{f}^*$ if appropriate, and prunes the search path.

The NO_EDGES special case

In step (1), CABOB checks whether the bid graph G has any edges ($|E| > 0$). If not, it accepts all of the remaining bids, updates $\tilde{f}^*$ if appropriate, and prunes the search path.

Preprocessing

Several preprocessing techniques have been proposed for search-based winner determination algorithms [Sandholm, 1999], and any of them could be used in conjunction with CABOB. However, in CABOB the search itself is fast, so we did not want to spend significant time preprocessing (since that could dwarf the search time). The only preprocessing that CABOB does is that as a bid B_x arrives, CABOB discards every bid B_y that B_x dominates ($p_x \geq p_y$ and $S_x \subseteq S_y$), and discards bid B_x if it is dominated by any earlier bid.

4 Bid Ordering Heuristics

In step (10) of CABOB, there are potentially a large number of bids on which CABOB could branch on. We developed several *bid ordering heuristics* for making this choice.[4]

- *Normalized Bid Price (NBP):* [Sandholm and Suri, 2000]. Branch on a bid with the highest $w_j = \frac{p_j}{(|S_j|)^\alpha}$. It was conjectured [Sandholm and Suri, 2000] that α slightly less than 0.5 would be best (since $\alpha = 0.5$ gives the best worst-case bound within a greedy algorithm [Lehmann *et al.*, 1999]), but we determined experimentally that $\alpha \in [0.8,\ 1]$ yields fastest performance.
- *Normalized Shadow Surplus (NSS):* The problem with NBP is that it treats each item as equally valuable. It could be modified to weight different items differently based on static prices that, e.g., the seller guesses before the auction. We propose a more sophisticated method where the items are weighted by their "values" *in the remaining subproblem*. We use the shadow price y_j from the remaining DUAL problem as a proxy for the worth of an item. We then branch on the bid whose price gives the highest surplus above the worth of the items (normalized by the worths so the surplus has to be greater if the bid uses valuable items): $w_j = \frac{p_j - \sum_{i \in S_j} y_i}{(\sum_{i \in S_j} y_i)^\alpha}$. Next we showed experimentally that the following modification to the normalization leads to faster performance: $w_j = \frac{p_j - \sum_{i \in S_j} y_i}{\log(\sum_{i \in S_j} y_i)}$. We call this scheme NSS.
- *Bid Graph Neighbors (BGN):* Branch on a bid with the largest number of neighbors in the bid graph G. The motivation is that this will allow CABOB to exclude the largest number of still eligible bids from consideration.
- *Number of Items (NI):* Branch on a bid with the largest number of items. The motivation is the same as in BGN.
- *One Bids (OB):* Branch on a bid whose x_j-value from LP is closest to 1. The idea is that the more of the bid is accepted in the LP, the more likely it is to be competitive.
- *Fractional Bids (FB):* Branch on a bid with x_j closest to 0.5. This strategy is advocated in the operations research literature [Wolsey, 1998]. The idea is that the LP is least sure about these bids, so it makes sense to resolve that uncertainty rather than to invest branching on bids about which the LP is "more certain". More often than not, the bids whose x_j values are close to 0 or 1 tend to get closer to those extreme values as search proceeds down a path, and in the end, LP will give an integer solution. Therefore those bids never end up being branched on.

[4]This corresponds to variable ordering. Choosing between the IN-branch ($x_j = 1$) and the OUT-branch ($x_j = 0$) first corresponds to value ordering. In the current version of CABOB, we always try the IN-branch first. The reason is that we try to include good bids early so as to find good solutions early. This enables more pruning through upper bounding. It also improves the anytime performance. CPLEX, on the other hand, uses value ordering as well in that it sometimes tries the OUT-branch first. In future research we plan to experiment with that option in CABOB as well.

4.1 Choosing Bid Ordering Heuristics Dynamically

We ran experiments on several distributions (discussed later) using each one of the heuristics as the primary heuristic, while using each of the other heuristics as a tie-breaker. We also tried using a third heuristic to break remaining ties, but that never helped. The best composite heuristic (OB+NSS) used OB first, and broke ties using NSS.

We noticed that on certain distributions, OB+NSS was best while on distributions where the bids included a large number of items, NSS alone was best. The selective superiority of the heuristics led us to the idea of choosing the bid ordering heuristic *dynamically based on the characteristics of the remaining subproblem*. We determined that the distinguishing characteristic between the distributions was LP density:

$$\text{density} = \frac{\text{number of nonzero coefficients in LP}}{\text{number of LP rows} \times \text{number of LP columns}}$$

OB+NSS was best when density was less than 0.25 and NSS was best otherwise. Intuitively, when the LP table is sparse, LP is good at "guessing" which bids to accept. When the table is dense, the LP makes poor guesses (most bids are accepted to a small extent). In those cases the price-based scheme NSS (that still uses the shadow prices from the LP) was better.

So, at every search node in CABOB, the density is computed, and the bid ordering scheme is chosen dynamically (OB+NSS if density is less than 0.25, NSS otherwise).

As a fundamentally different bid ordering methodology, we observe that stochastic local search—or any other approximate algorithm for the problem—could be used to come up with a good solution fast, and then that solution could be forced to be the left branch (IN-branch) of CABOB (with the "most sure" bids nearest the root) so as to give CABOB a more global form of guidance in bid ordering.

5 Design Philosophy of CABOB vs. CPLEX

We benchmarked CABOB against a general-purpose integer programming package, CPLEX 7.0. It was recently shown [Andersson *et al.*, 2000] that CPLEX 6.5 is faster (or comparable) in determining winners in combinatorial auctions than are the first-generation special-purpose search algorithms [Sandholm, 1999; Fujishima *et al.*, 1999]. CPLEX 7.0 is about 1.6 times faster than CPLEX 6.5, so when we compare CABOB against CPLEX 7.0, to our knowledge, we are comparing it against the state-of-the-art.

There are some fundamental differences between CABOB and CPLEX that we want to explain to put the experiments in context. CPLEX uses best-bound search (A*) [Wolsey, 1998] which requires exponential space (it also has an option to force depth-first search, but that makes CPLEX somewhat slower), while CABOB uses depth-first branch-and-bound (DFBnB) which runs in linear space. Thus, on some hard problems, CPLEX ran out of virtual memory. In our experiments we only show cases where CPLEX was able to run in RAM. DFBnB puts CABOB at a disadvantage when it comes to reaching the optimal solution fast since it does not allow CABOB to explore the most promising leaves of the search tree first. At the same time, we believe that the memory issue make A* unusable for combinatorial auctions in practice. Like CABOB, CPLEX uses LP to obtain upper bounds.

CPLEX uses a "presolver" to manipulate the LP table algebraically [Wolsey, 1998] to reduce it before search. In the experiments, we ran CABOB without any presolving. Naturally, that could be added to CABOB as a preprocessing step.

Put together, everything else being equal, CPLEX should find an optimal solution and prove optimality faster than DFBnB, but one would expect the anytime behavior to be worse.

6 Experimental Setup

We tested CABOB and CPLEX on the common combinatorial auction benchmarks distributions: those of [Sandholm, 1999], and the CATS distributions [Leyton-Brown *et al.*, 2000]. In addition, we tested them on new distributions.

The distributions from [Sandholm, 1999] are:

- **Random:** For each bid, pick the number of items randomly from $1, 2, ..., m$. Randomly choose that many items without replacement. Pick a price from $[0, 1]$.
- **Weighted random:** As above, but pick the price between 0 and the number of items in the bid.
- **Uniform:** Draw the same number of randomly chosen items for each bid. Pick the prices from $[0, 1]$.
- **Decay:** Give the bid one random item. Then repeatedly add a new random item with probability α until an item is not added or the bid includes all m items. Pick the price between 0 and the number of items in the bid. In the tests we used $\alpha = 0.75$ since the graphs in [Sandholm, 1999] show that this setting leads to the hardest instances on average (at least for that algorithm).

We tested the algorithms on all of the CATS distributions: **paths**, **regions**, **matching**, **scheduling**, and **arbitrary**. For each one of them, we used the default parameters in the CATS instance generators, and varied the number of bids.

We also tested the algorithms on the following new benchmark distributions:

- **Bounded:** For each bid, draw the number of items randomly between a lower bound and an upper bound. Randomly include that many distinct items in the bid. Pick the price between 0 and the number of items in the bid.
- **Components:** A number of independent problems, each from the uniform distribution.

We generate bids so no two bids have the same set of items. The experiments were conducted on a 933 MHz Pentium III PC with 512MB RAM. Each point in each plot is the median run time for 100 instances. CABOB and CPLEX both use the default LP solver that comes with CPLEX (dual simplex). CABOB and CPLEX were tested on the same instances.

7 Experimental Results

On the random distribution (Fig 1 left), CABOB is faster than CPLEX and the difference grows with the number of bids. The preprocessor of CABOB eliminates a large number of bids. CABOB always resorted to search while CPLEX's presolve+LP solved the problem without search on 47% of the instances. On the weighted random distribution (Fig 1 right), the performance of CABOB and CPLEX is almost identical. However, they achieve this very differently. With 2,000 bids, CPLEX's presolve+LP solves the problem 95% of the time while CABOB resorts to search 88% of the time.

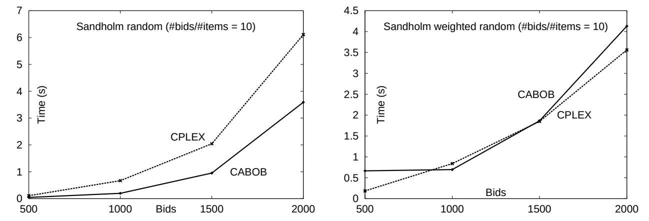

Figure 1: *Run times on the* **random** *and* **weighted random** *distributions.*

Interestingly, on the decay distribution (Fig. 2 left)—which is perhaps the most realistic distribution of those of [Sandholm, 1999], and was reported to be difficult for the earlier winner determination algorithms—both algorithms solve the problem using LP in almost all cases. CPLEX goes to search 2% of the time while CABOB resorts to search only 1% of the time. CABOB is faster than CPLEX, mainly because CPLEX uses presolve while CABOB does not.

On the uniform distribution (Fig. 2 right), both algorithms resort to search. The speeds are comparable, but CPLEX is faster. For the first-generation winner determination algorithms [Sandholm, 1999; Fujishima *et al.*, 1999], the instances with small numbers of items per bid were much harder than instances with long bids. For both CABOB and CPLEX, complexity is almost invariant to the number of items per bid, except that complexity *drops* significantly as the bids include less than 5 items each! This is because LP can handle cases with short bids well, both in terms of upper bounding and finding integer solutions. (If each bid contains only *one* item, LP *always* finds an integer solution).

The components distribution demonstrates the power of CABOB's decomposition technique and pruning across com-

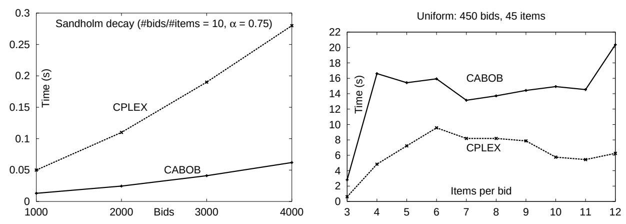

Figure 2: *Run times on the* **decay** *and* **uniform** *distributions.*

ponents. CABOB's run-time increases linearly with the number of components while CPLEX's time is exponential (Fig. 3). Already at 5 components, CPLEX ran out of virtual memory. The same performance would be observed even if there were a "glue" bid that included items from each component, since CABOB would identify that bid as an articulation bid.

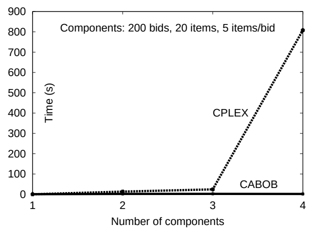

Figure 3: *Run times on the* **components** *distribution.*

On the bounded distribution (Fig. 4)—which is a more realistic version of the uniform distribution—the relative performance of CABOB and CPLEX depended on the bounds. For short bids, CPLEX was somewhat faster, but the *relative* speed difference decreased with the number of bids. For long bids, CABOB was much faster (mainly due to checking for completeness of the bid graph G), and the difference grew dramatically with the number of bids.

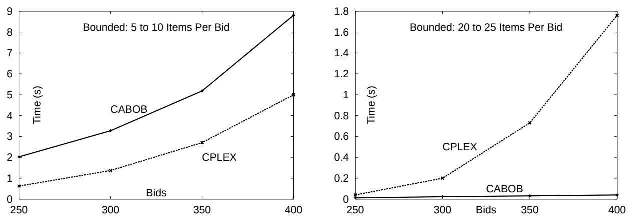

Figure 4: *Run times on the* **bounded** *distribution.*

Surprisingly, the CATS distributions were very easy. LP solved them in almost all cases. Interestingly, even the rare cases where the algorithms resorted to search disappeared as the number of bids *increased* (except for CATS arbitrary where the complexity did not vary much with the number of bids). As Fig. 5 shows, CABOB was faster than CPLEX (mainly because the preprocessor discards a large number of bids). The difference grows with the number of bids.

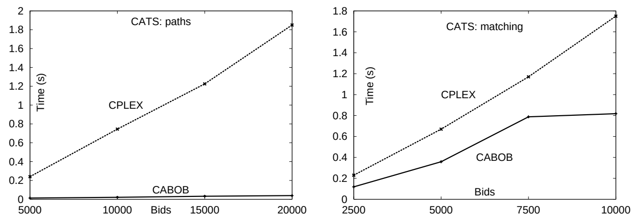

Figure 5: *Run times on CATS* **paths** *and* **matching**.

7.1 Anytime Performance

As expected from their designs, CABOB has better anytime performance than CPLEX. Fig. 6 shows a run that is typical in the sense of anytime performance, but which was carefully selected so that CABOB and CPLEX take equal time to prove that an optimal solution has been reached. CABOB dominates CPLEX throughout, and finds the optimal solution in 40% of the time it takes CPLEX to find it.

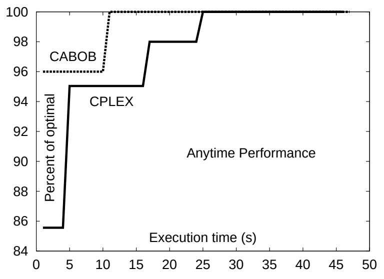

Figure 6: *Solution quality on the bounded distribution, reported by each algorithm once per second.*

8 Random Restarts

Random restarts have been widely used in local search algorithms, but recently they have been shown to speed up tree search algorithms as well [Gomes *et al.*, 1998]. We conjectured that random restarts, combined with randomized bid ordering, could avoid the perils of unlucky bid ordering. To see whether we could improve CABOB using random restarts, we implemented the random restarts methods that are best (to our knowledge) and improved them further to try to capitalize on the special properties of the problem.

We implemented the following restart strategies:

- *Double:* Double the execution time between restarts.
- *Constant:* Restart after every δ backtracks [Gomes *et al.*, 1998].
- *Luby-Sinclair-Zuckerman:* [Luby *et al.*, 1993] showed that the constant scheme above is optimal if δ is tailored to the run-time distribution, which is, unfortunately, usually not known in practice. Therefore, they constructed a scheme that suffers only an asymptotically logarithmic time penalty, independent of the run-time distribution. In the scheme, each run time is a power of 2. Each time a pair of runs of the same length has been executed, a run time of twice that length is immediately executed: $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \ldots$.

We implemented the following bid ordering techniques to use with the restart strategies:

- *Random:* Randomly pick a remaining bid.
- *Boltzmann:* Pick a bid with probability $p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$, where $q_i = x_i + \frac{w_j}{LPUB}$. The value w_j is from the NSS bid ordering heuristic, and $LPUB$ is the objective function value from LP. Higher values of T result in more randomness.
- *Bound:* Each bid whose x_j value is within a bound b of the highest x_j value is equally probable.

We tried every bid ordering with every restart strategy, and varied the initial time allotment and the parameters δ, T, and b. CABOB was always faster than CABOB with restarts!

It turns out that this is not just a facet of our restart schemes or parameters settings. Random restarts tend to lead to speedup when the run-time distribution has a heavy tail [Gomes *et al.*, 1998]. We decided to test whether CABOB exhibits heavy-tailed run-times on the winner determination problem. We chose the distribution on which CABOB's run-time varied the most so as to increase the chance of finding a heavy tail. This was the uniform distribution with 5 items per bid. If a distribution has a heavy tail, the variance and usually also the mean are unbounded [Gomes *et al.*, 1998]. As can be seen in Fig. 7, our mean and variance are not only bounded, but constant. This means that the run-time distribution does not have a heavy tail. This explains our negative results with restarts, and suggests that random restarts are not a fruitful avenue for future improvement in this setting.

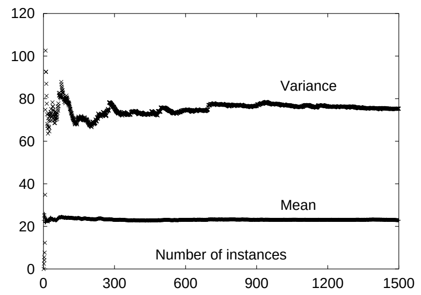

Figure 7: *The mean and variance of CABOB's search time as a function of the number of instances in the sample.*

9 Conclusions and Future Research

Combinatorial auctions where bidders can bid on bundles of items can lead to more economical allocations, but determining the winners is $\mathcal{NP}$-complete and inapproximable. We presented CABOB, a sophisticated search algorithm for the problem. It uses decomposition techniques, upper and lower bounding (also across components), a host of structural observations, elaborate and dynamically chosen bid ordering heuristics, and other techniques to increase speed—especially on problems with different types of special structure, which we expect to be common in real combinatorial auctions. Experiments against the fastest prior algorithm, CPLEX 7.0, show that CABOB is usually faster, never drastically slower, and in many cases drastically faster. Overall, this makes CABOB, to our knowledge, the currently fastest optimal algorithm for the problem. CABOB's search runs in linear space while CPLEX takes exponential space. CABOB also has significantly better anytime behavior than CPLEX.

We also uncovered interesting aspects of the problem itself. First, the problems with short bids that were hard for the first-generation of specialized algorithms are easy. Second, almost all of the CATS distributions are easy, and become easier with more bids. Third, we tested a number of random restart strategies, and showed that random restarts do not help on this problem because the run-time distribution does not have a heavy tail (at least not for CABOB).

We are currently working not only on designing faster algorithms for winner determination in combinatorial auctions, but also on winner determination in combinatorial reverse auctions and exchanges [Sandholm *et al.*, 2001], as well as in combinatorial markets with additional side constraints [Sandholm and Suri, 2001]. We are also developing methods for intelligent, selective elicitation of combinatorial bids from the market participants [Conen and Sandholm, 2001].

References

[Andersson *et al.*, 2000] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In *ICMAS*, pages 39–46.

[Conen and Sandholm, 2001] Wolfram Conen and Tuomas Sandholm. Minimal preference elicitation in combinatorial auctions. In *IJCAI-2001 Workshop on Economic Agents, Models, and Mechanisms*.

[de Vries and Vohra, 2000] Sven de Vries and Rakesh Vohra. Combinatorial auctions: A survey. August 28th.

[Fujishima *et al.*, 1999] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI*, pages 548–553.

[Gomes *et al.*, 1998] Carla Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *AAAI*.

[Hoos and Boutilier, 2000] Holger Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In *AAAI*, pages 22–29.

[Lehmann *et al.*, 1999] Daniel Lehmann, Lidian Ita O'Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *ACM Conference on Electronic Commerce* , pages 96–102.

[Leyton-Brown *et al.*, 2000] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce* , pages 66–76.

[Luby *et al.*, 1993] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47:173–180.

[Nisan, 2000] Noam Nisan. Bidding and allocation in combinatorial auctions. In *ACM Conference on Electronic Commerce* , pages 1–12.

[Rassenti *et al.*, 1982] S J Rassenti, V L Smith, and R L Bulfin. A combinatorial auction mechanism for airport time slot allocation. *Bell J. of Economics*, 13:402–417.

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

[Sandholm and Suri, 2000] Tuomas Sandholm and Subhash Suri. Improved algorithms for optimal winner determination in combinatorial auctions and generalizations. In *AAAI*, pages 90–97.

[Sandholm and Suri, 2001] Tuomas Sandholm and Subhash Suri. Side constraints and non-price attributes in combinatorial markets. In *IJCAI-2001 Workshop on Distributed Constraint Reasoning*.

[Sandholm *et al.*, 2001] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. Winner determination in combinatorial auction generalizations. In *AGENTS Workshop on Agent-Based Approaches to B2B*.

[Sandholm, 1993] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *AAAI*, p. 256–262.

[Sandholm, 1999] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI*, pages 542–547, 1999. First appeared as Washington Univ., Dept. of Computer Science, WUCS-99-01, Jan. 28th.

[Sandholm, 2000] Tuomas Sandholm. eMediator: A next generation electronic commerce server. In *AGENTS*, pages 73–96, 2000. Early version: AAAI-99 Workshop on AI in Electronic Commerce, Orlando, FL, pp. 46–55, July 1999, and Washington University, St. Louis, Dept. of Computer Science WU-CS-99-02, Jan. 1999.

[Wolsey, 1998] Laurence Wolsey. *Integer Programming*. John Wiley.