MANAGEMENT SCIENCE
Vol. 51, No. 3, March 2005, pp. 374–390
ISSN 0025-1909 | EISSN 1526-5501 | 05 | 5103 | 0374

informs

DOI 10.1287/mnsc.1040.0336

CABOB: A Fast Optimal Algorithm for Winner Determination in Combinatorial Auctions

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

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

Andrew Gilpin, David Levine
CombineNet, Inc., Fifteen 27th Street, Pittsburgh, Pennsylvania 15222
{agilpin@combinenet.com, dlevine@combinenet.com}

Combinatorial auctions where bidders can bid on bundles of items can lead to more economically efficient allocations, but determining the winners is $\mathcal{NP}$-complete and inapproximable. We present CABOB, a sophisticated optimal 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. CABOB attempts to capture structure in any instance without making assumptions about the instance distribution. Experiments against the fastest prior algorithm, CPLEX 8.0, show that CABOB is often faster, seldom drastically slower, and in many cases drastically faster—especially in cases with structure. CABOB's search runs in linear space and has significantly better anytime performance than CPLEX.

We also uncover interesting aspects of the problem itself. First, problems with short bids, which were hard for the first generation of specialized algorithms, are easy. Second, almost all of the CATS distributions are easy, and the run time is virtually unaffected by the number of goods. Third, we test several random restart strategies, showing that they do not help on this problem—the run-time distribution does not have a heavy tail.

Key words: auction; combinatorial auction; winner determination; winner-determination algorithm; search; branch and bound; MIP; anytime algorithm; branching heuristics; dynamically chosen heuristic; bounding across components; random restart

History: Accepted by G. Anandalingam and S. Raghavan, special issue editors; received June 4, 2002. This paper was with the authors 5 months for 2 revisions.

1. Introduction

In many auctions, a bidder's valuation for a combination of distinguishable items that are for sale is not the sum of the individual items' valuations—it can be more, or less. This is often the case, for example, in electricity markets, equities trading, bandwidth auctions (McMillan 1994, McAfee and McMillan 1996), transportation exchanges (Sandholm 1993, 2000, 1991; Net Exchange, Inc. 2001), pollution rights auctions, auctions for airport landing slots (Rassenti et al. 1982), supply chains (Sandholm 2002b, Walsh et al. 2000, Babaioff and Nisan 2001), and auctions for carrier-of-last-resort responsibilities for universal services (Kelly and Steinberg 2000). *Combinatorial auctions* where bidders can bid on bundles of items (Rassenti et al. 1982, Sandholm 1993) allow bidders to express *complementarity* among the items (and, with a rich enough bidding language, also *substitutability* among the items) (Sandholm 2002a, b; Fujishima et al. 1999; Nisan 2000; Hoos and Boutilier 2001). Perhaps the most well-known combinatorial auction is the Federal Communications Commission's planned combinatorial auction for spectrum licenses. Less publicized, in business-to-business commerce, billions of dollars worth of combinatorial auctions are conducted annually by vendors such as CombineNet, Manhattan Associates, NetExchange, and Trade Extensions.

Due to the expressiveness that combinatorial auctions offer to the bidders, such auctions tend to yield more economically efficient allocations of the items because bidders do not get stuck with partial bundles that are of low value to them. In the academic literature this has been demonstrated, for example, in airport landing slot allocation (Rassenti et al. 1982) and in markets for trucking tasks (Sandholm 1993, Net Exchange, Inc. 2001).

However, determining the winners in a combinatorial auction is computationally complex, and there has been a surge of research into addressing that problem. Three fundamentally different approaches have

been taken: (1) designing algorithms that provably find an optimal solution but are slow on some problem instances (Sandholm 2002a; Fujishima et al. 1999; Sandholm and Suri 2003; Andersson et al. 2000; de Vries and Vohra 2003; Gonen and Lehmann 2000; Lehmann and Gonen 2001; Leyton-Brown et al. 2000b; van Hoesel and Müller 2001; Balas and Yu 1986; Babel and Tinhofer 1990; Babel 1991; Balas and Xue 1991, 1996; Nemhauser and Sigismondi 1992; Mannino and Sassano 1994; Pardalos and Desai 1991; Loukakis and Tsouros 1983), (2) designing algorithms that are provably fast but fail to find an optimal (or even close to optimal) solution to some problem instances (Lehmann et al. 2002, Hoos and Boutilier 2000, Zurel and Nisan 2001, Anandalingam et al. 2002), and (3) restricting the bundles on which bids can be submitted so severely that the remaining problem can be solved optimally and provably fast (Rothkopf et al. 1998, Sandholm and Suri 2003, Tennenholtz 2000, Penn and Tennenholtz 2000, van Hoesel and Müller 2001, Lehmann et al. 2005).

The third approach suffers from economic inefficiencies and exposure problems similar to those of noncombinatorial auctions because the bidders might not be allowed to bid on the bundles they desire. The second approach suffers from wasting economic efficiency whenever a suboptimal solution is settled on, and because the winner-determination problem is inapproximable (Sandholm 2002a), no fast algorithm can guarantee that its solution is even close to optimal. Furthermore, suboptimal winner determination generally compromises the incentive properties of the auction (Nisan and Ronen 2000; Sandholm 2002b, a). Due to these reasons, we focus on the first approach—with the understanding that on some problem instances, any algorithm within this approach will take a long time. We present an optimal search algorithm for the winner-determination problem, and show that the algorithm is fast in practice. The algorithm is the newest development in our multiperson R&D effort within this approach, which has been ongoing since 1997 (Sandholm 2002a, Sandholm and Suri 2003).

The rest of the paper is organized as follows. We formally define the problem in §2 and present our core algorithm in §3. Section 4 discusses bid-ordering heuristics. Experiments are presented in §§5–7. Random restart strategies are discussed in §8. Section 9 presents conclusions and future research directions.

2. The Winner-Determination Problem

In this section we formally define the winner-determination problem. In the following sections, we will present our algorithm for optimally solving this 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 $p_j \in R$, $p_j \geq 0$, 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:[1]

$$\begin{aligned} \max \ & \sum_{j=1}^{n} p_j x_j \\ \text{s.t.} \ & \sum_{j \mid i \in S_j} x_j \leq 1, \quad \forall i \in \{1..m\} \\ & x_j \in \{0, 1\}. \end{aligned}$$

This problem is $\mathcal{NP}$-complete (Rothkopf et al. 1998, Karp 1972). The problem cannot even be approximated to a ratio of $n^{1-\epsilon}$ in polynomial time, for any fixed $\epsilon > 0$ (unless $\mathcal{ZPP} = \mathcal{NP}$) (Sandholm 2002a).[2]

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.

$$\begin{aligned} \text{LP} \quad \max \ & \sum_{j=1}^{n} p_j x_j \\ & \sum_{j \mid i \in S_j} x_j \leq 1, \quad \forall i \in \{1..m\} \\ & x_j \geq 0 \\ & x_j \in \mathbb{R} \\ \text{DUAL} \quad \min \ & \sum_{i=1}^{m} y_i \\ & \sum_{i \in S_j} y_i \geq p_j, \quad \forall j \in \{1..n\} \\ & y_i \geq 0 \\ & y_i \in \mathbb{R}. \end{aligned}$$

In this continuous setting, the *shadow price* y_i gives the price for each individual item i. In the binary case, individual items cannot generally be given prices, but

[1] If there is no free disposal (auctioneer is not willing to keep any of the items, and bidders are not willing to take extra items), an equality is used in place of the inequality. We focus on the problem with free disposal, as does the bulk of the literature on combinatorial auctions.

[2] This was recently proven using an approximation-preserving reduction from the MAX-CLIQUE problem, which is known to be inapproximable (unless $\mathcal{ZPP} = \mathcal{NP}$) (Håstad 1999). For a review of the approximability of special cases of combinatorial auctions, see Sandholm (2002a).

each y_i value from DUAL gives an upper bound on the price of item i.[3]

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 in the BOB algorithm (Sandholm and Suri 2003). However, BOB was not implemented, and experimental results were never published. CABOB incorporates many of the techniques proposed in BOB and a host of additional ones. All of them have been implemented, and experimental results are presented.

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 stored in a global variable $\tilde{f}^*$. Initially, $\tilde{f}^* = 0$.

We maintain a conflict graph structure that we call the *bid graph*, denoted by G. The nodes of the graph correspond to bids that are still available to be appended to the search path, that is, bids that do not include any items that have already been allocated. So, the number of vertices $|V| \leq n$. Two vertices in G share an edge whenever the corresponding bids share items.[4,5,6] 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 reinserted into G when backtracking, the edges are also reinserted.

[3] In general the solution to the DUAL is not unique, that is, several shadow price vectors y are optimal. According to the concerns that the DUAL captures, any optimal shadow price vector is equally justified. When we use shadow prices in CABOB (as will be described later in the paper), we use that optimal shadow price vector that the linear programming algorithm that solves the DUAL returns.

[4] Because 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 bids containing a large number of items it sometimes took almost as much time as the search).

[5] One potential problem is that the bid graph could be prohibitively large to store if the problem were huge. One could address this by generating a bid graph only when the graph size is below some threshold. Then, if the size falls below the threshold later on some search path, one could generate the bid graph then, for the remaining subtree of the search tree. If the graph is sparse, the graph size tends to be small enough to construct the graph at the root. On the other hand, if the graph is dense, the average node degree is large. Then every IN branch will kill off a large part of the (implicit) bid graph, so the search would quickly get to a point of having a small explicitly constructable bid graph.

[6] Conflict graphs have been used to speed up optimization problems before in different ways. In Atamtürk et al. (2000), efficient data structures for conflict graphs were presented when the goal was cut generation. In our setting, the purpose is neighbor finding and decomposition. The data structures in Atamtürk et al. (2000) do not support these operations efficiently.

The following pseudocode of CABOB makes calls to several special cases that will be introduced later. For readability, we only show how the values are updated, and 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. $CABOB(G, g, \text{MIN})$

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

13.d. $f_i^* \leftarrow CABOB(G_i, g + g_i', \text{MIN} - g_i')$
13.e. $\text{MIN} \leftarrow \text{MIN} + (\tilde{f}^* - f_{old}^*)$
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 at more length.

3.1. Upper Bounding

In Step (3), CABOB uses an upper bound on the revenue that 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 (in our implementation of CABOB, we used the LP solver that comes with CPLEX rather than write our own LP solver), whose objective function value gives an upper bound. By LP we mean the LP that is defined by the remaining bids, that is, bids that are still in G.

CABOB does not make copies of the LP table, but rather incrementally deletes (reinserts) columns corresponding to the bids being deleted (reinserted) in 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.[7] 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. We made this design choice because CABOB uses the solutions from the LP and the DUAL for several other purposes beyond upper bounding (such as for the INTEGER special case, for bid ordering, and for random restart methods—as we will discuss later).

Our experiments showed that using LP as the upper-bounding method led to significantly faster completion times of the search algorithm than using any of the other upper-bounding methods proposed for combinatorial auction winner determination before (Sandholm 2002a, Fujishima et al. 1999, Sandholm and Suri 2003). This is likely due to better bounding, better bid ordering, and the effect of the INTEGER special case, described below. The time taken to solve the LP at every node was greater than the per-node time with the other upper-bounding methods, but the reduction in tree size amply paid for that.

[7] 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.

3.2. 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 frequently, contrary to our expectations), 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. Sufficient conditions under which the LP provides an integer-valued solution are described in Gul and Stacchetti (1999), Nisan (2000), and Bikhchandani and Ostroy (2002).

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. Consider the following auction with six bids and six items. The bids are given in Table 1.

Table 1 **A Combinatorial Auction Instance Illustrating That Bids Whose LP Relaxation Values Are Integral May Not Be Excluded from Subsequent Search**

Bid	Items	Price
A	{1, 2}	2
B	{2, 3}	2
C	{1, 3, 4}	2
D	{3, 4, 5}	2
E	{5, 6}	4.5
F	{6}	3

This combinatorial auction corresponds to the following mathematical program:

$$\begin{aligned} \max \quad & 2A + 2B + 2C + 2D + 4.5E + 3F \\ \text{such that} \quad & A + C \le 1 \\ & A + B \le 1 \\ & B + C + D \le 1 \\ & C + D \le 1 \\ & D + E \le 1 \\ & E + F \le 1. \end{aligned}$$

Table 2 gives the unique optimal values for each of the decision variables for both the combinatorial auction (that is, the integer program (IP)) and the corresponding LP.

Table 2 Optimal IP and LP Values of the Combinatorial Auction Instance Given in Table 1

Bid	IP	LP
A	1	0.5
B	0	0.5
C	0	0.5
D	1	0
E	0	1
F	1	0

Note that in the optimal LP solution, bid E has a value of 1, but in the optimal IP solution E has a value of 0. Also note that bids D and F both have a value of 0 in the optimal LP solution, but have values of 1 in the optimal IP solution. In summary, even though the LP unambiguously suggests that bids D and F be rejected and bid E be accepted, any one of these actions would compromise optimality of the algorithm.

3.3. 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. Rounding can be effective for this purpose (Hoffman and Padberg 1993); 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 $x_j > \frac{1}{2}$ 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 solution.[8] This method gives a lower bound because the solution it obtains is feasible. (If an item is contained in more than one bid, at most one of those bids can have $x_j > \frac{1}{2}$ in the LP, so at most one of those bids will be inserted into the lower-bound solution in the first phase of the lower-bound construction. In the second phase, feasibility is maintained by only considering for insertion bids that do not share items with bids already in the lower-bound solution.)

Our experiments showed that the lower-bounding technique did not help very much. One reason is that the search algorithm's left branch (where all bids on the path are winning) provides a lower bound itself. Therefore, the potential advantage from explicit lower bounding at a node comes from the lower bound being found early without having to invest the costly solving of several LPs (one per search node) that are involved in traversing the left branch.

[8] Note that inserting a bid into the lower-bound solution does not mean inserting the bid into the search path. Also, it does not mean that the bid is removed from G.

In the future we plan to try other (additional) lower-bounding techniques within CABOB beyond rounding techniques—such as stochastic local search (Hoos and Boutilier 2000). Additional lower bounds cannot hurt in terms of the number of search nodes because the search algorithm can use the highest (that is, the best) of the lower bounds. However, there is a trade-off between reducing the size of the search tree via sophisticated lower-bounding techniques and reducing the per-node time by using only quick lower bounding.

3.4. Exploiting Decomposition

Decomposition techniques are another powerful tool to incorporate into search algorithms. The idea is to partition the bids into sets (aka connected components) so that no bid from one set shares items with any bid from any other set. The winner determination can then be conducted in each set separately.

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. Because search time is super-linear 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 speed-up.

3.5. Upper and Lower Bounding *Across Components*

Upper- and lower-bounding techniques are common in tree-search algorithms, and CABOB uses upper and lower bounding as discussed above. However, in addition to common 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

because now the bar of when search is useful has been raised.[9] 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.

3.6. 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 (2003).

The scheme of always branching on an articulation bid, if one exists, is often at odds with price-based bid-ordering schemes, discussed later. It has been proven that no scheme from the articulation-based family dominates any scheme from the price-based family, or vice versa, in general (Sandholm and Suri 2003). 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 at each search tree node 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.

3.7. The COMPLETE Special Case

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

3.8. 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.

3.9. The ALL_NEIGHBORS Special Case

In Step (10.b), CABOB checks whether the bid to branch on, B_k, neighbors all other bids in G. If so, none of the other bids can be accepted. Therefore, CABOB never actually proceeds to the branch where B_k is accepted, but simply tries to include B_k, and updates $\tilde{f}^*$ if appropriate. CABOB then proceeds to the branch where B_k is rejected. This saves the time of removing all the vertices and edges from G and then immediately reinserting them.

3.10. Preprocessing

Several preprocessing techniques have been proposed for search-based winner determination algorithms (Sandholm 2002a), 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 (because 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.[10] Because this can be done incrementally as bids arrive, and therefore does not factor into the winner-determination time after the auction closes, this time (which is negligible anyway) is not included in the experiments.

[9] This causes the MIN values to stay nonnegative throughout the tree.

[10] This preprocessor (also used in Fujishima et al. 1999) is a special case of Preprocessor 2 presented in Sandholm (2002a). It corresponds to Preprocessor 2 with the search depth confined to one.

4. Bid-Ordering Heuristics

In Step (10) of CABOB, there are potentially a large number of bids on which CABOB could branch. We developed several *bid-ordering heuristics* for making this choice.[11,12] They are geared toward finding good solutions early. This has two advantages. First, if the algorithm has to be stopped before reaching an optimal solution (or before having proven that the best solution found so far is optimal), a good solution is available. This is often called the *anytime* aspect of a search algorithm. Second, seeing good solutions early on reduces total run time because more of the search tree gets pruned—mainly due to enhanced upper bounding.

We conducted detailed experiments with the following bid-ordering heuristics:

• *Normalized Bid Price* (*NBP*) (*Sandholm and Suri 2003*): Branch on a bid with the highest

$$w_j = \frac{p_j}{(|S_j|)^\alpha}. \quad (1)$$

It was hypothesized (Sandholm and Suri 2003) that α slightly less than $\frac{1}{2}$ would be best (because $\alpha = \frac{1}{2}$ gives the best worst-case bound within a greedy algorithm, Lehmann et al. 2002), 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, for example, 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 value of an item. We then branch on the bid whose price gives the highest surplus above the value of the items[13] (normalized by the values so the surplus has to be greater if the bid uses valuable items):

$$w_j = \frac{p_j - \sum_{i \in S_j} y_i}{(\sum_{i \in S_j} y_i)^\alpha}. \quad (2)$$

Next we showed experimentally that the following modification to the normalization leads to faster performance:

$$w_j = \frac{p_j - \sum_{i \in S_j} y_i}{\log(\sum_{i \in S_j} y_i)}. \quad (3)$$

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*: 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*: Branch on a bid with x_j closest to $\frac{1}{2}$. This strategy has been widely advocated in the operations research literature (e.g., Wolsey 1998, p. 99). 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.

We ran experiments on several distributions (discussed later), using all possible pairs of these bid-ordering heuristics for primary bid selection and tiebreaking, respectively. We also tried using a third heuristic to break remaining ties, but that never helped. The speed difference between CABOB with

[11] 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.

[12] Bid-ordering heuristics have also been developed for winner determination in multiunit combinatorial auctions (Sandholm and Suri 2003, Leyton-Brown et al. 2000b, Gonen and Lehmann 2000, Lehmann and Gonen 2001) and combinatorial exchanges (Sandholm and Suri 2003).

[13] A related approach is *column generation* (Barnhart et al. 1998). It seems best suited when the problem contains a huge number of variables (columns). (Typically these variables are implicitly defined, such as sequences of flight legs in crew scheduling, and thus there can be exponentially many of them in the explicit input size.) Because of the infeasibility of dealing with such a huge set of variables, the column generation method works with only a small subset of variables, and brings into the LP basis new columns on demand. In CABOB, we are typically never faced with problems where the number of bids is too large to explicitly work with. The complexity of winner determination does not seem to arise from a large number of bids, but rather from the structure of the bids. However, the idea of using shadow prices from the LP dual to choose which bid to next branch on does have some resemblance to column generation's method of identifying which bid/column to bring into the LP. One important difference, however, may be that while column generation only adds a bid/column to the LP basis, CABOB actually branches on that bid explicitly, setting it to 1 or 0.

the best heuristics and CABOB with the worst heuristics was greater than two orders of magnitude. The best composite heuristic (OB+NSS) used OB first, and broke ties using NSS.

4.1. Choosing Bid-Ordering Heuristics *Dynamically*

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 a distinguishing characteristic between the distributions was LP density:

$$\text{density} = \frac{\text{number of nonzero coefficients in LP}}{\text{number of LP rows} \times \text{number of LP columns}}. \quad (4)$$

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). This is the bid-ordering scheme that we use in the experiments presented later in this paper.

As a fundamentally different bid-ordering methodology, we observe that stochastic local search—or any other approximate algorithm for the winner-determination 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's search tree. Committing (as an initial guess) to the entire set of accepted bids from the approximate solution in this way would give CABOB a more global form of guidance in bid ordering than conducting bid ordering on a per-bid basis. To refine this method further, CABOB could take hints (for example from the approximation algorithm) as to how "surely" different bids that are accepted in the approximate solution should be accepted in the optimal solution. In the left branch (IN-branch) of CABOB, the "most sure" bids should then be assigned closest to the root of the search tree, because bids near the root will be the last ones to be backtracked in the search. This ordering will allow good solutions to be found early, and (mainly due to upper bounding) avoids unnecessary search later on.

5. Design Philosophy of CABOB vs. CPLEX

We benchmarked CABOB against a general-purpose integer programming package, CPLEX 8.0. 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 2002a, Fujishima et al. 1999). CPLEX 7.0 is reported to be about 60% faster than CPLEX 6.5 (ILOG Inc. 2000), and CPLEX 7.1 is as much as 21% faster than CPLEX 7.0, based on a few comparisons that we performed. CPLEX 8.0 performance differs from CPLEX 7.1: It is better in some instances and worse in others. Overall, we have found it to be about the same. Therefore, to our knowledge, CPLEX 8.0 is the fastest prior optimal algorithm for the problem. Furthermore, it was recently shown that in combinatorial auction winner determination, CPLEX performs favorably even against incomplete search algorithms—such as stochastic local search—that do not generally find the optimal solution (Schuurmans et al. 2001). Therefore, when we compare CABOB against CPLEX 8.0, to our knowledge we are comparing it against the state-of-the-art general-purpose solver.

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 (Wolsey 1998),[14,15] which requires exponential space (CPLEX also has an option to force depth-first search, but that makes the search slower), while CABOB uses depth-first branch-and-bound (DFBnB), which runs in linear space.[16] Thus, on many harder problems, CPLEX ran out of virtual memory and

[14] Best-bound search is identical to A* search (Hart et al. 1968, Russell and Norvig 1995) if the next node to expand is always chosen to be the node with the greatest $f = g + h$. However, to avoid solving a node's LP when the node is first seen (and only solving the LP when the node comes up for expansion), sometimes a node's parent's LP value (or some refinement thereof) is used in practice as a proxy for the node's h value.

[15] Actually, CPLEX uses a slightly enhanced version of best-bound search, where the search continues down the current path if the current path is almost as promising (in terms of the value of $f = g + h$) as the most promising node on the open list (ILOG Inc. 2002). This bias (how close the current node's f value has to be to the best f value on the open list) is controlled by the `backtrack` parameter within CPLEX. We tried different settings, and concluded that the default setting is very competitive. Therefore, we left the `backtrack` parameter at its default setting in the experiments that we present in this paper.

[16] CABOB could be converted to use another search strategy. We considered SMA* (Russell 1992) and recursive best-first search (Korf 1993). However, we decided against SMA* because it would require keeping copies of the bid graph G (one for each leaf of the search tree), which would require frequent slow copying and would use a large amount of memory. We decided against recursive

stopped before it found the optimal solution. In our experiments we show only cases where CPLEX was able to run in RAM. Everything else being equal, DFBnB should put CABOB at a disadvantage when it comes to reaching the optimal solution quickly because it does not allow CABOB to explore the most promising leaves of the search tree first (we show in the experiments that CABOB was often faster nevertheless). At the same time, we believe that the memory issue can make best-bound search unusable for combinatorial auctions of realistic sizes in practice. Like CABOB, CPLEX uses LP to obtain upper bounds.

CPLEX uses a presolver to manipulate the LP table algebraically (Wolsey 1998, ILOG Inc. 2002) to reduce it before search.[17] In the experiments, we ran CABOB without any such presolving.

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

6. Experimental Setup

We tested CABOB and CPLEX on the common combinatorial auction benchmarks distributions: those presented in Sandholm (2002a), and the *Combinatorial Auction Test Suite* (*CATS*) distributions (Leyton-Brown et al. 2000a). In addition, we tested them on new distributions.

The distributions from Sandholm (2002a) follow.

- *Random*(m, n): Generate n bids as follows. For each bid, (1) pick the number of items randomly from $\{1, 2, \ldots, m\}$, (2) randomly choose that many items without replacement from $\{1, 2, \ldots, m\}$, and (3) pick a price from a uniform distribution on $[0, 1]$.[18]
- *Weighted random*(m, n): As above, but pick the price between 0 and the number of items in the bid.
- *Uniform*(m, n, λ): Generate n bids as follows. For each bid, (1) randomly choose λ items without replacement from $\{1, 2, \ldots, m\}$, and (2) pick a price from a uniform distribution on $[0, 1]$.
- *Decay*(m, n, α): Generate n bids as follows. Give the bid one random item from $\{1, 2, \ldots, m\}$. Then repeatedly add a new random item from $\{1, 2, \ldots, m\}$ (without replacement) 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 $\alpha = 0.75$ because the graphs in Sandholm (2002a) show that this setting leads to the hardest instances on average (at least for that algorithm).

We tested the algorithms on all of the combinatorial auction benchmark distributions available in the CATS suite: *paths*, *regions*, *matching*, *scheduling*, and *arbitrary*.[19] For each one of these, we used the default parameters in the CATS instance generators, and varied the number of bids n and the number of items m.

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

- *Bounded*$(m, n, \underline{\lambda}, \bar{\lambda})$: Generate n bids as follows. For each bid, (1) draw the number of items λ randomly between a lower bound $\underline{\lambda}$ and an upper bound $\bar{\lambda}$, (2) randomly include λ distinct items from $\{1, 2, \ldots, m\}$ in the bid, and (3) pick the price from a uniform distribution on $[0, \lambda]$. This distribution is a more realistic variant of the uniform distribution in the sense that it includes bids with different numbers of items.
- *Components*(m, n, λ, C): Generate C problems (which are independent in the sense that each one has its own set of m items), each from the distribution Uniform(m, n, λ). This distribution models, for example, auctions where the items are specific to geographical regions (such as spectrum licenses), and each bidder—such as a radio station—is only interested in licenses within her own metropolitan area.[20]

We generate bids so no two bids have the same set of items. The experiments were conducted on a 2.8 GHz Pentium IV PC with 4 GB RAM. Each point in each plot is the mean run time for 100 instances. CABOB and CPLEX both use the default LP solver that comes with CPLEX (dual simplex) (ILOG Inc. 2002). CABOB and CPLEX were tested on the same problem instances.

best-first search because it leads to significant amounts of redundant search on problems where the edge costs of the search tree are real numbers.

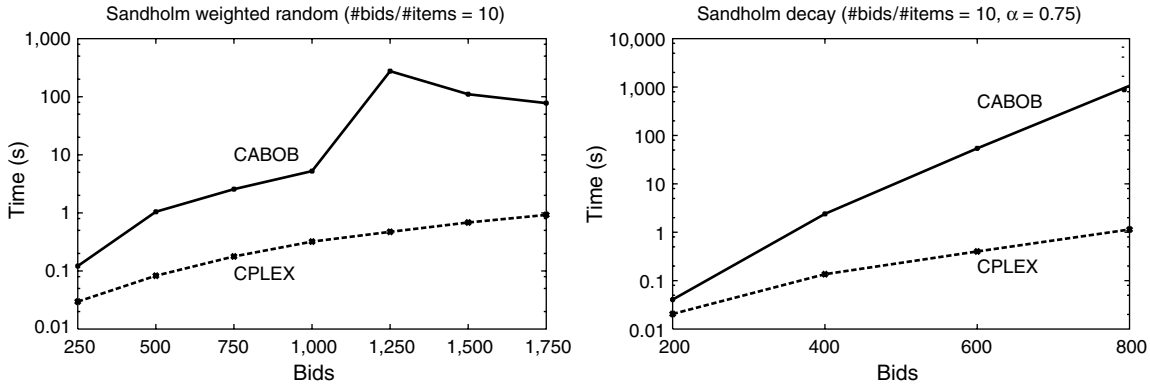
[17] We do not know of any way to do the preprocessing steps that CPLEX uses incrementally as bids arrive. Therefore, in the experiments we include CPLEX's preprocessing time in CPLEX's overall run time. However, CPLEX's preprocessing time on these problems is usually negligible anyway.

[18] The random distribution is a particularly well-suited input for a bid dominance preprocessor such as that used in CABOB. Because the resulting problem is drastically different from the original problem, we do not include any experimental results for this distribution, as any comparison would mostly reflect the effect of the preprocessor rather than search performance.

[19] CATS also includes distributions that are not benchmarks for combinatorial auctions, but rather for multiunit combinatorial auctions, a generalized problem.

[20] In many spectrum auctions, there are additionally some bidders that want licenses across metropolitan areas—such as mobile phone network operators that want national coverage. While the components distribution does not model this, CABOB's articulation and decomposition techniques apply to this setting as well. Once enough of those broader bids have been branched on, the remaining bid graph will have an articulation bid. CABOB will branch on that bid, causing the problem to decompose. The components distribution is mainly used as an extreme example of how well our decomposition strategy can boost performance in practice.

Figure 1 Run Times on the *Weighted Random* and *Decay* Distributions



7. Experimental Results

The weighted random distribution (Figure 1, left) is easy for both algorithms. The algorithms achieve their performance very differently. With 1,000 bids, CPLEX's presolve + LP solves the problem 95% of the time while CABOB resorts to search 88% of the time. Above 1,000 bids, CABOB's performance varies greatly. Most of the time CABOB solves the problem quickly, but occasionally it must resort to search, which results in longer average performance.

The decay distribution (Figure 1, right) was significantly harder for both algorithms.[21] Both algorithms resort to search. CPLEX was significantly faster than CABOB due to its best-bound search strategy (as opposed to CABOB's depth-first branch and bound), its use of cutting planes to reduce the LP polytope (ILOG Inc. 2002), and its presolver.

The uniform distribution (Figure 2) was even harder than the decay distribution (roughly equally hard for CABOB, and significantly harder for CPLEX). Both algorithms resorted to search. The figure on the left shows how the algorithms scale as the number of bids increases, and the figure on the right shows how the algorithms scale as the number of items per bid varies. The speeds are comparable, but CPLEX is faster. For the first-generation winner-determination algorithms (Sandholm 2002a, Fujishima et al. 1999), the instances with small numbers of items per bid were much harder than instances with long bids. (This was because a search path can contain a large number of short bids, but only a small number of long bids. Therefore, the search depth—ignoring any pruning—is greater with short bids.) For both CABOB and CPLEX, complexity is quite insensitive to the number of items per bid, except that complexity *drops* significantly as the bids include less than five items each! This is because LP tends to 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.)

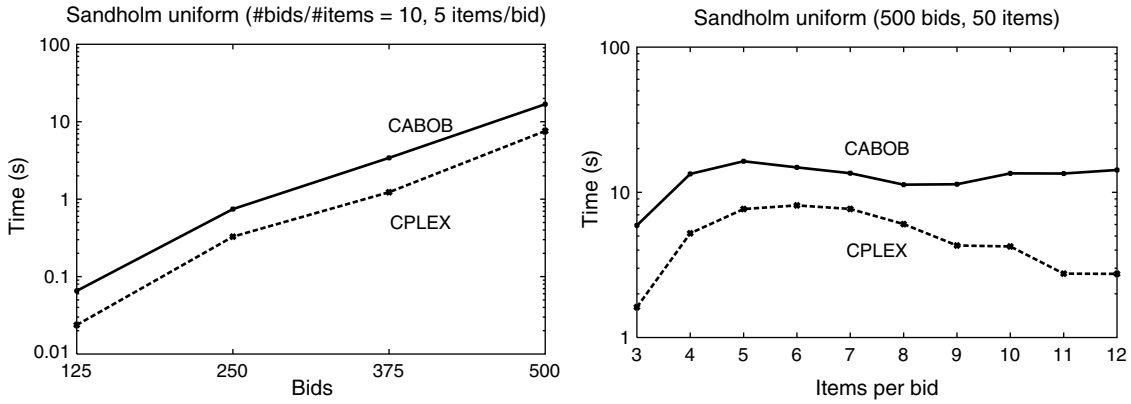
While the problem instances from the distributions discussed above exhibit no structure, the components distribution demonstrates the performance of the algorithms on structured instances. In particular, it shows the power of CABOB's decomposition technique and pruning across components in an extreme example. CABOB's run time increases linearly with the number of components. The run time of CPLEX increases exponentially (Figure 3). (The same performance would be observed even if there were a single "glue" bid that included items from each component, because CABOB would identify that bid as an articulation bid.) While the decomposition technique helps drastically when the problem is decomposable, it also increases the time per search node. On problems where decompositions are rare, this leads to a net increase in run time. For example, on the decay distribution, CABOB's run time would be reduced by a third if the decomposition technique were turned off.

On the bounded distribution (Figure 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 for small bids (mainly due to checking for completeness of the bid graph G), but as the number of bids continues to grow, a complete bid graph occurs less often.

The CATS distributions were surprisingly easy for both solvers. They are extremely easy, except for the *scheduling* distribution, which is moderately difficult. (These observations hold at least for the default settings of the CATS instance generator parameters.)

[21] The decay distribution is significantly harder, both for CPLEX and CABOB, than we originally presented in the IJCAI-01 version of this paper, because in the old version there was an error in the problem instance generator. The authors thank Mattias Tenhunen and Fredrik Ygge for raising suspicion that the performance in the original graph seemed too good. This led us to find the error in the decay distribution instance generator.

Figure 2 Run Times on the *Uniform* Distribution



This suggests that more realistic instance distributions of the winner determination problem are in fact *easier* than the more random, unstructured ones.

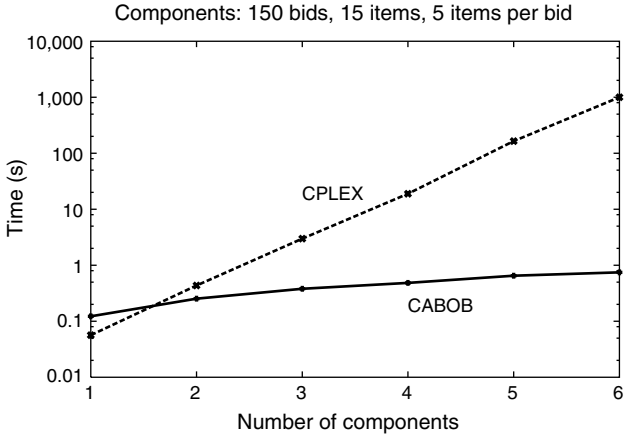
There are five distributions in CATS: *paths, regions, scheduling, arbitrary*, and *matching*. Experimental results from these distributions appear below. For each distribution, two graphs are shown: one where the number of bids is varied, and one where the number of items is varied. Interestingly, on the CATS distributions, the run time of the algorithms is virtually unaffected by the number of items in the auction (for a fixed number of bids).

The *paths* distribution was one of the easy distributions. As Figure 5 shows, both algorithms scale very well. CABOB is more than an order of magnitude faster than CPLEX.

The *regions* distribution was more difficult for both algorithms, but nevertheless easy (Figure 6). The *scheduling* distribution is of medium difficulty as Figure 7 shows. It is the hardest of the CATS distributions. Both algorithms scale very well. On these distributions, CPLEX is faster than CABOB.

The *arbitrary* and *matching* distributions (Figures 8 and 9) were also easy. Interestingly, as the number of bids increases beyond a certain point (for a fixed number of items), CABOB's run time *decreases*. The reason is that, on these distributions, once there is a sufficient number of bids, the LP finds an integral solution at the root of the search tree, so CABOB does not need to resort to search.

Figure 3 Run Times on the *Components* Distribution



7.1. Anytime Performance

The anytime performance of a search algorithm is important so that if the algorithm happens to take more time than is available, it can be terminated with a reasonably good solution available. As expected from their respective designs, CABOB has better anytime performance than CPLEX. Figure 10 illustrates this phenomenon. Each curve is averaged over 100 problem instances. CABOB achieves close to optimal solution quality significantly faster than CPLEX even on this problem distribution (uniform distribution), on which CPLEX completes its search significantly faster overall (see Figure 2).

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 (searching large parts of the state space that do not contain an optimal solution). 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).

Figure 4 **Run Times on the *Bounded* Distribution**

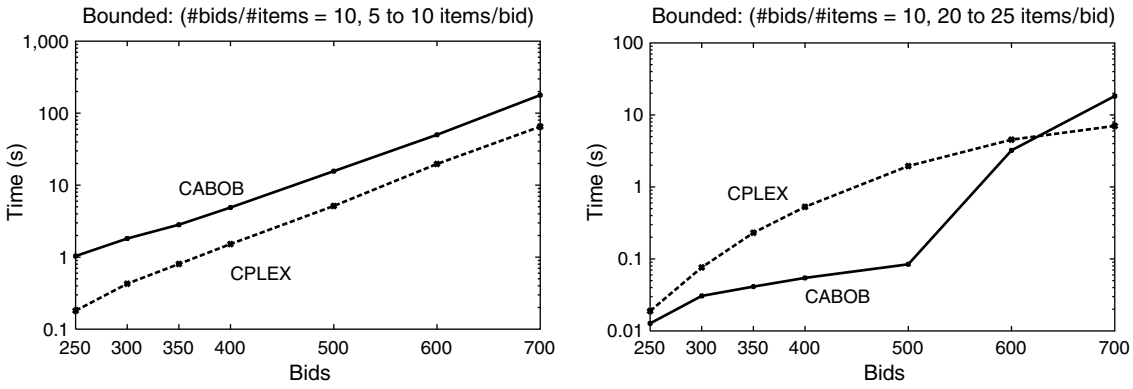


Figure 5 **Run Times on CATS *Paths***

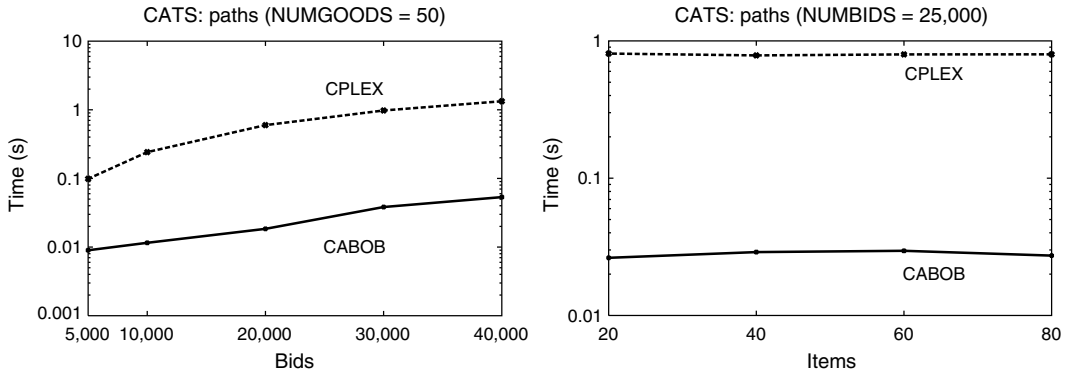


Figure 6 **Run Times on CATS *Regions***

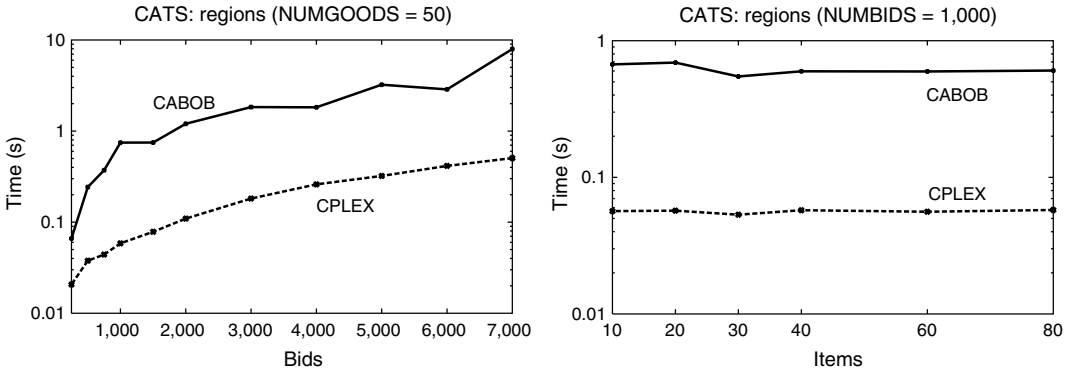


Figure 7 **Run Times on CATS *Scheduling***

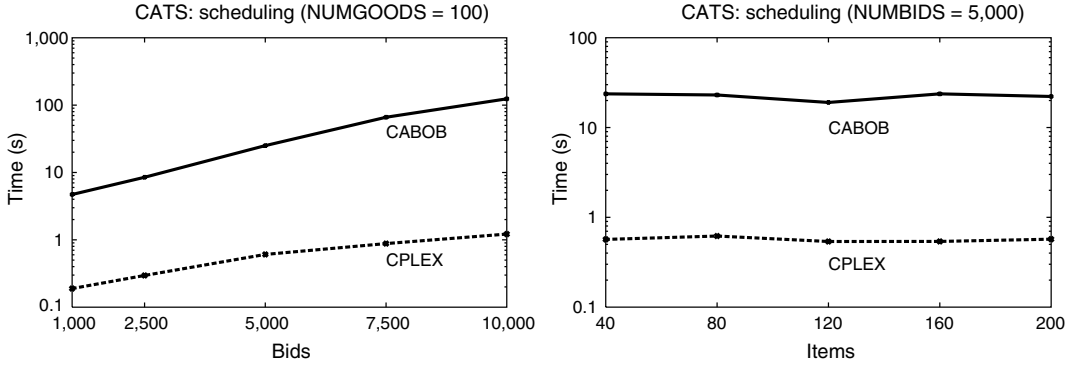
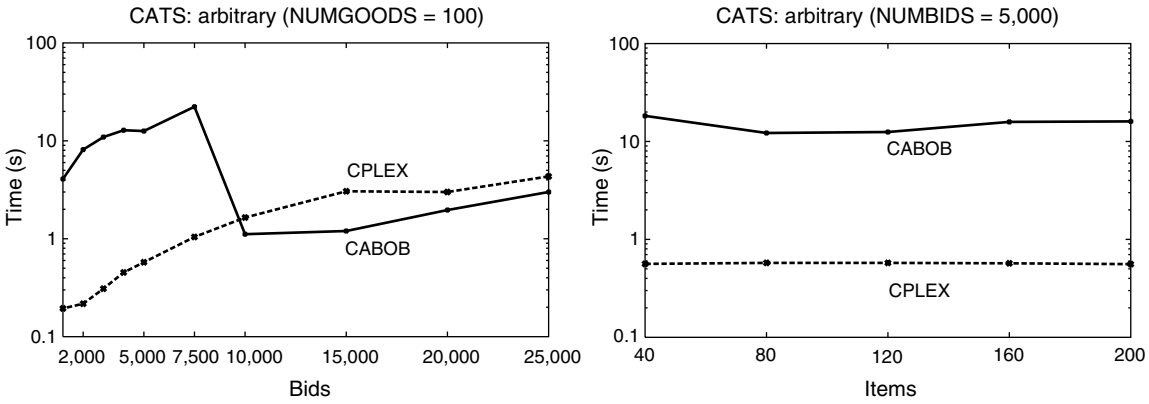


Figure 8 Run Times on CATS *Arbitrary*



• *Luby-Sinclair-Zuckerman*: (Luby et al. 1993).[22] Luby et al. 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, ...

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

$$p_j = \frac{e^{q_j/T}}{\sum_k e^{q_k/T}}, \tag{5}$$

where higher values of T result in more randomness, and q_j is a measure of how promising bid j is. We used

$$q_j = x_j + \frac{w_j}{LPUB}, \tag{6}$$

where x_i is the decision variable from the remaining LP, w_j is from the Normalized Shadow Surplus (NSS) bid-ordering heuristic (Equation (3)), and *LPUB* (the objective function value of the remaining LP) is used for normalization. In other words, this definition of q_j uses the intuitions from both of the best bid-ordering heuristics (One Bids (OB), and Normalized Shadow Surplus (NSS)) in determining how promising a bid is.

• *Bound*: Each bid whose x_j value is within a bound b of the highest x_j value is equally probable.

[22] Actually, the results of Luby et al. (1993) are for independent runs. Our runs are not really independent because $\tilde{f}^*$ is carried over from completed runs when starting a new run. This method is better than independent runs, but might cause the results of Luby et al. not to hold.

We tried every bid ordering with every restart strategy, and varied the initial time allotment and the parameters δ, T, and b. The results of our experiments showed that CABOB was always faster than CABOB with restarts.

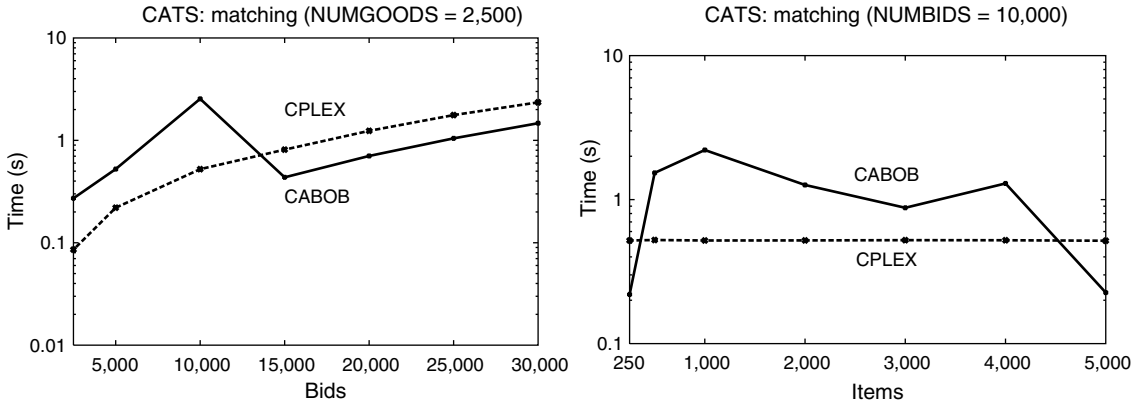
It turns out that this is not just a facet of our restart schemes or parameters settings. Random restarts tend to lead to speed-up when the run-time distribution has a heavy tail (Gomes et al. 1998). We decided to test whether CABOB exhibits heavy-tailed run times on the winner-determination problem. We chose a distribution on which CABOB's run time varied greatly, so as to increase the chance of finding a heavy tail. This was the uniform distribution with five 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 Figure 11, our mean and variance are not only bounded, but constant. This means that the run-time distribution does not have a heavy tail. This suggests that random restarts are not a fruitful avenue for future improvement in this setting.

Our findings suggest several alternative hypotheses about the benefits of restarts: (1) Restarts do not help in optimization problems; they can only help in constraint satisfaction problems. (2) Restarts help on some optimization problems as well, but combinatorial auctions are within a subclass of optimization problems on which restarts do not help, and (3) restarts help only if the search algorithm (variable and value-ordering heuristics, upper and lower-bounding techniques, decomposition techniques, etc.) is bad, and CABOB is not bad. Disambiguating among these hypotheses is an interesting direction for future research on restart strategies in general, not just in the context of combinatorial auctions.

9. Conclusions and Future Research

Combinatorial auctions where bidders can bid on bundles of items can lead to more economically efficient allocations, but determining the winners

Figure 9 **Run Times on CATS *Matching***



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. CABOB attempts to automatically identify different forms of structure, and to exploit such structure when it exists. Experiments against the fastest prior algorithm, CPLEX 8.0, show that CABOB is often faster, seldom drastically slower, and in many cases drastically faster—especially in cases with structure. CABOB's search runs in linear space, while CPLEX takes exponential space, and often runs out of virtual memory. CABOB also has significantly better anytime performance than CPLEX. Based on these observations, we feel that CABOB contains many search techniques that are useful for winner determination in combinatorial auctions, and potentially for other optimization problems.

We also uncovered interesting aspects of the problem itself. First, problems with short bids, which were hard for the first generation of specialized algorithms, are easy. Second, almost all of the CATS distributions are easy, and the run time is virtually unaffected by the number of items in the auction. On two of the CATS distributions, CABOB's run time decreases as the number of bids increases beyond a certain point. Third, we tested a number of random restart strategies, and showed that random restarts do not help on this problem—the run-time distribution does not have a heavy tail (at least not for CABOB).

We hope that the ideas presented in CABOB will facilitate the development of even faster winner-determination algorithms in the future. 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 and Suri 2003, Sandholm et al. 2002, Kothari et al.

Figure 10 **Anytime Performance: Average Solution Quality on the Uniform Distribution (7 Items per Bid, 50 Items, 500 Bids), Reported by Each Algorithm Once per Second**

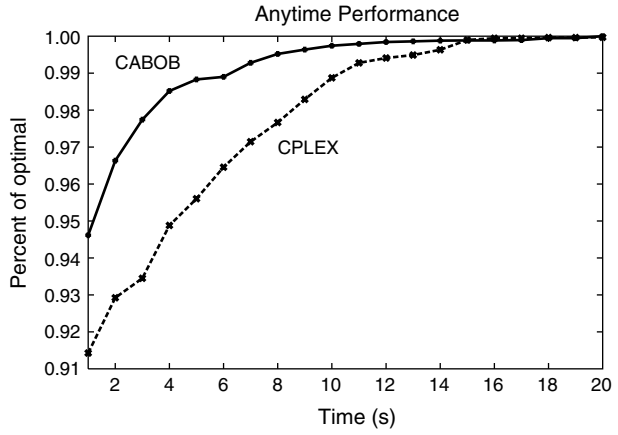
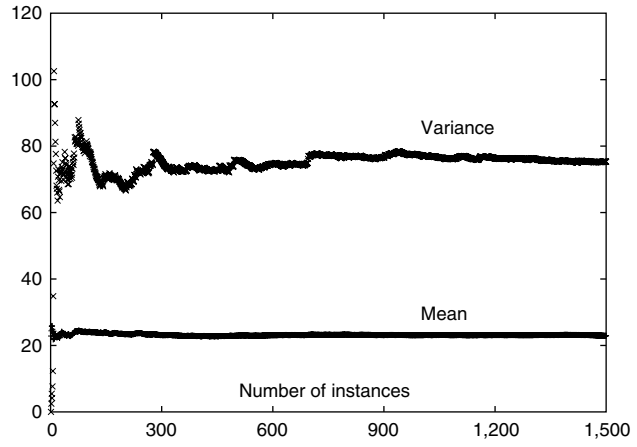


Figure 11 **The Mean and Variance of CABOB's Search Time as a Function of the Number of Instances in the Sample**



2003), as well as in combinatorial markets with additional side constraints (Sandholm and Suri 2001).

Beyond winner determination, there are several other interesting research directions within combinatorial auctions. Designing mechanisms (rules) for iterative combinatorial auctions is one such direction (Sandholm 1993; DeMartini et al. 1999; Parkes 1999; Parkes and Ungar 2000a, b; Wurman and Wellman 2000; Bikhchandani et al. 2001; Ausubel and Milgrom 2002). A more recent, more general direction is to supplement the auctioneer with an *elicitor* that selectively elicits bids from the bidders in order to determine a good allocation of items to bidders without requiring the bidders to bid on all combinations of items (Conen and Sandholm 2001a, b, 2002b, a; Smith et al. 2002; Zinkevich et al. 2003; Blum et al. 2004; Santi et al. 2004; Lahaie and Parkes 2004). While the communication complexity of combinatorial auctions is exponential in the worst case (Nisan and Segal 2003), in practice only a vanishingly small fraction of the bidders' private valuation information needs to be elicited in order to determine the optimal allocation (and incentive compatible payments to be made by the bidders) (Hudson and Sandholm 2004). Another interesting direction for future research in combinatorial auctions is to design a proxy bidder agent that not only bids on the user's behalf, but also computes the user's (and his competitors') valuations based on an optimization model of how each party would use the bundles of items that he might win (Larson and Sandholm 2001b, a, 2004b, a).

Acknowledgments

This work was funded by, and conducted at, CombineNet, Inc., Fifteen 27th St., Pittsburgh, PA 15222. A short early version of this paper appeared in the *Proceedings of the International Joint Conference on Artificial Intelligence* (*IJCAI*), Seattle, WA, 1102–1108, August 2001.

References

Anandalingam, G., R. Kwon, L. Ungar. 2002. An efficient approximation algorithm for combinatorial auctions. Working paper, Center for Electronic Markets and Enterprises, University of Maryland, College Park, MD.

Andersson, Arne, Mattias Tenhunen, Fredrik Ygge. 2000. Integer programming for combinatorial auction winner determination. *Proc. Fourth Internat. Conf. Multi-Agent Systems* (*ICMAS*). IEEE Computer Society, Boston, MA, 39–46.

Atamtürk, A., G. L. Nemhauser, M. W. P. Savelsbergh. 2000. Conflict graphs in solving integer programming problems. *Eur. J. Oper. Res.* **121** 40–55.

Ausubel, Lawrence M., Paul Milgrom. 2002. Ascending auctions with package bidding. *Frontiers Theoret. Econom.* **1**(1).

Babaioff, Moshe, Noam Nisan. 2001. Concurrent auctions across the supply chain. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*) *Tampa, FL*. ACM, New York, 1–10.

Babel, Luitpold. 1991. Finding maximal cliques in arbitrary and special graphs. *Computing* **46** 321–341.

Babel, Luitpold, Gottfried Tinhofer. 1990. A branch and bound algorithm for the maximum weighted clique problem. *ZOR—Methods Models Oper. Res.* **34** 207–217.

Balas, Egon, Jue Xue. 1991. Minimum weighted coloring of triangulated graphs, with application to maximum weighted vertex packing and clique finding in arbitrary graphs. *SIAM J. Comput.* **20**(2) 209–221.

Balas, Egon, Jue Xue. 1996. Weighted and unweighted maximum clique algorithms with upper bonds from fractional coloring. *Algorithmica* **15** 397–412.

Balas, Egon, Chang Sung Yu. 1986. Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.* **15**(4) 1054–1068.

Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, P. H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* **46** 316–329.

Bikhchandani, Sushil, Joseph M. Ostroy. 2002. The package assignment model. *J. Econom. Theory* **107** 377–406.

Bikhchandani, Sushil, Sven de Vries, James Schummer, Rakesh V. Vohra. 2002. Linear programming and Vickrey auctions. Working paper.

Blum, Avrim, Jeffrey Jackson, Tuomas Sandholm, Martin Zinkevich. 2004. Preference elicitation and query learning. *J. Mach. Learning Res.* **5** 649–667.

Conen, Wolfram, Tuomas Sandholm. 2001a. Preference elicitation in combinatorial auctions: Extended abstract. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*), *Tampa, FL*. ACM, New York, 256–259.

Conen, Wolfram, Tuomas Sandholm. 2001b. Minimal preference elicitation in combinatorial auctions. *Proc. Internat. Joint Conf. Artificial Intelligence*, (*IJCAI*), *IJCAI*-2001 *Workshop on Economic Agents, Models, and Mechanisms, Seattle, WA*. 71–80.

Conen, Wolfram, Tuomas Sandholm. 2002a. Differential-revelation VCG mechanisms for combinatorial auctions. *AAMAS-02 Workshop Agent-Mediated Electronic Commerce (AMEC)*, Bologna, Italy. *Lecture Notes in Computer Science*, No. 2531. Springer, Berlin, Germany.

Conen, Wolfram, Tuomas Sandholm. 2002b. Partial-revelation VCG mechanism for combinatorial auctions. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Edmonton, Canada*. AAAI Press, Menlo Park, CA, 367–372.

DeMartini, Christine, Anthony M. Kwasnica, John O. Ledyard, David Porter. 1999. A new and improved design for multiobject iterative auctions. Technical report 1054, California Institute of Technology, Social Science, Pasadena, CA.

de Vries, Sven, Rakesh Vohra. 2003. Combinatorial auctions: A survey. *INFORMS J. Comput.* **15**(3) 284–309.

Fujishima, Yuzo, Kevin Leyton-Brown, Yoav Shoham. 1999. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. *Proc. Sixteenth Internat. Joint Conf. Artificial Intelligence* (*IJCAI*), *Stockholm, Sweden*. 548–553.

Gomes, Carla, Bart Selman, Henry Kautz. 1998. Boosting combinatorial search through randomization. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Madison, WI*. AAAI Press, Menlo Park, CA.

Gonen, Rica, Daniel Lehmann. 2000. Optimal solutions for multiunit combinatorial auctions: Branch and bound heuristics. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*), *Minneapolis, MN*. ACM, New York, 13–20.

Gul, Faruk, Ennio Stacchetti. 1999. Walrasian equilibrium with gross substitutes. *J. Econom. Theory* **87** 95–124.

Hart, Peter E., Nils J. Nilsson, Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Sci. Cybernetics* **4**(2) 100–107.

Håstad, Johan. 1999. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.* **182** 105–142.

Hoffman, Karla L., Manfred Padberg. 1993. Solving airline crew-scheduling problems by branch-and-cut. *Management Sci.* **39**(6) 657–682.

Hoos, Holger, Craig Boutilier. 2000. Solving combinatorial auctions using stochastic local search. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Austin, TX*. AAAI Press, Menlo Park, CA, 22–29.

Hoos, Holger, Craig Boutilier. 2001. Bidding languages for combinatorial auctions. *Proc. Seventeenth Internat. Joint Conf. Artificial Intelligence* (*IJCAI*), *Seattle, WA*. 1211–1217.

Hudson, Benoit, Tuomas Sandholm. 2004. Effectiveness of query types and policies for preference elicitation in combinatorial auctions. *Internat. Conf. Autonomous Agents and Multi-Agent Systems (AAMAS)*, New York, 386–393.

ILOG Inc. 2000. CPLEX presentation. INFORMS, San Antonio, TX.

ILOG Inc. 2002. *CPLEX 8.0 User's Manual*.

Karp, Richard M. 1972. Reducibility among combinatorial problems. Raymond E. Miller, James W. Thatcher, eds. *Complexity of Computer Computations*. Plenum Press, New York, 85–103.

Kelly, Frank, Richard Steinberg. 2000. A combinatorial auction with multiple winners for universal services. *Management Sci.* **46**(4) 586–596.

Korf, Richard E. 1993. Linear-space best-first search. *Artificial Intelligence* **62**(1) 41–78.

Kothari, Anshul, Tuomas Sandholm, Subhash Suri. 2003. Solving combinatorial exchanges: Optimality via a few partial bids. *Proc. ACM Conf. Electronic Commerce (ACM-EC), San Diego, CA*. ACM, New York, 236–237.

Lahaie, Sebastién, David Parkes. 2004. Applying learning algorithms to preference elicitation. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*). ACM, New York.

Larson, Kate, Tuomas Sandholm. 2001a. Computationally limited agents in auctions. *AGENTS-01 Workshop of Agents for B2B*. Montreal, Canada, 27–34.

Larson, Kate, Tuomas Sandholm. 2001b. Costly valuation computation in auctions. *Theoretical Aspects of Rationality and Knowledge (TARK VIII)*. Siena, Italy, 169–182.

Larson, Kate, Tuomas Sandholm. 2004a. Designing auctions for deliberative agents. *Internat. Conf. Autonomous Agents Multi-Agent Systems (AAMAS), Workshop Agent-Mediated Electronic Commerce (AMEC)*. New York, 225–238.

Larson, Kate, Tuomas Sandholm. 2004b. Experiments on deliberation equilibria in auctions. *Internat. Conf. Autonomous Agents Multi-Agent Systems (AAMAS)*. New York, 394–401.

Lehmann, Benny, Daniel Lehmann, Noam Nisan. 2005. Combinatorial auctions with decreasing marginal utilities. *Games Econom. Behavior* Forthcoming.

Lehmann, Daniel, Rica Gonen. 2001. Linear programming helps solving large multi-unit combinatorial auction. *Electronic Market Design Workshop*. Maastricht, The Netherlands.

Lehmann, Daniel, Lidian Ita O'Callaghan, Yoav Shoham. 2002. Truth revelation in rapid, approximately efficient combinatorial auctions. *J. ACM* **49**(5) 577–602.

Leyton-Brown, Kevin, Mark Pearson, Yoav Shoham. 2000a. Towards a universal test suite for combinatorial auction algorithms. *Proc. ACM Conf. Electronic Commerce (ACM-EC), Minneapolis, MN*. ACM, New York, 66–76.

Leyton-Brown, Kevin, Moshe Tennenholtz, Yoav Shoham. 2000b. An algorithm for multi-unit combinatorial auctions. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Austin, TX*. AAAI Press, Menlo Park, CA.

Loukakis, E., C. Tsouros. 1983. An algorithm for the maximum internally stable set in a weighted graph. *Internat. J. Comput. Math.* **13** 117–129.

Luby, Michael, Alistair Sinclair, David Zuckerman. 1993. Optimal speedup of Las Vegas algorithms. *Inform. Processing Lett.* **47** 173–180.

Mannino, Carlo, Antonio Sassano. 1994. An exact algorithm for the maximum stable set problem. *Comput. Optim. Appl.* **3** 242–258.

McAfee, R. Preston, John McMillan. 1996. Analyzing the airwaves auction. *J. Econom. Perspect.* **10**(1) 159–175.

McMillan, John. 1994. Selling spectrum rights. *J. Econom. Perspect.* **8**(3) 145–162.

Nemhauser, George L., G. Sigismondi. 1992. A strong cutting plane/branch-and-bound algorithm for node packing. *J. Oper. Res. Soc.* **43**(5) 443–457.

Net Exchange, Inc. 2001. Market architecture: Improving markets by enhancing choice, http://www.nex.com/docs/nexst.pdf (May).

Nisan, Noam. 2000. Bidding and allocation in combinatorial auctions. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*), *Minneapolis, MN*. ACM, New York, 1–12.

Nisan, Noam, Amir Ronen. 2000. Computationally feasible VCG mechanisms. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*), *Minneapolis, MN*. ACM, New York, 242–252.

Nisan, Noam, Ilya Segal. 2003. The communication requirements of efficient allocations and supporting prices. *J. Econom. Theory* Forthcoming.

Pardalos, Panos M., Nisha Desai. 1991. An algorithm for finding a maximum weighted independent set in an arbitrary graph. *Internat. J. Comput. Math.* **38** 163–175.

Parkes, David C. 1999. iBundle: An efficient ascending price bundle auction. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*), *Denver, CO*. ACM, New York, 148–157.

Parkes, David C., Lyle Ungar. 2000a. Iterative combinatorial auctions: Theory and practice. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Austin, TX*. AAAI Press, Menlo Park, CA, 74–81.

Parkes, David C., Lyle Ungar. 2000b. Preventing strategic manipulation in iterative auctions: Proxy-agents and price-adjustment. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Austin, TX*. AAAI Press, Menlo Park, CA, 82–89.

Penn, Michal, Moshe Tennenholtz. 2000. Constrained multi-object auctions and *b*-matching. *Inform. Processing Lett.* **75**(1–2) 29–34.

Rassenti, Stephen J., Vernon L. Smith, R. L. Bulfin. 1982. A combinatorial auction mechanism for airport time slot allocation. *Bell J. Econom.* **13** 402–417.

Rothkopf, Michael H., Aleksandar Pekeč, Ronald M. Harstad. 1998. Computationally manageable combinatorial auctions. *Management Sci.* **44**(8) 1131–1147.

Russell, Stuart. 1992. Efficient memory-bounded search methods. *Proc. Eur. Conf. Artificial Intelligence* (*ECAI*), Vienna, Austria, 1–5.

Russell, Stuart, Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ.

Sandholm, Tuomas. 1991. A strategy for decreasing the total transportation costs among area-distributed transportation centers. *Nordic Operations Analysis in Cooperation (NOAS): OR in Business*. Turku School of Economics, Finland.

Sandholm, Tuomas. 1993. An implementation of the contract net protocol based on marginal cost calculations. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Washington, D.C.* AAAI Press, Menlo Park, CA, 256–262.

Sandholm, Tuomas. 2000. Issues in computational Vickrey auctions. *Internat. J. Electronic Commerce* **4**(3) 107–129.

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

Sandholm, Tuomas. 2002b. eMediator: A next generation electronic commerce server. *Comput. Intelligence* **18**(4) 656–676.

Sandholm, Tuomas, Subhash Suri. 2001. Side constraints and non-price attributes in markets. *IJCAI-2001 Workshop on Distributed Constraint Reasoning*. Seattle, WA, 55–61.

Sandholm, Tuomas, Subhash Suri. 2003. BOB: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence* **145** 33–58.

Sandholm, Tuomas, Subhash Suri, Andrew Gilpin, David Levine. 2002. Winner determination in combinatorial auction generalizations. *Internat. Conf. Autonomous Agents and Multi-Agent Systems (AAMAS)*. Bologna, Italy, 69–76.

Santi, Paolo, Vincent Conitzer, Tuomas Sandholm. 2004. Towards a characterization of polynomial preference elicitation with value queries in combinatorial auctions. *Conf. Learning Theory (COLT)*. Banff, Alberta, Canada, 1–16.

Schuurmans, Dale, Finnegan Southey, Robert Holte. 2001. The exponentiated subgradient algorithm for heuristic Boolean programming. *Proc. Seventeenth Internat. Joint Conf. Artificial Intelligence* (*IJCAI*), Seattle, WA, 334–341.

Smith, Trey, Tuomas Sandholm, Reid Simmons. 2002. Constructing and clearing combinatorial exchanges using preference elicitation. *AAAI-02 Workshop on Preferences in AI and CP: Symbolic Approaches. Edmonton, Alberta, Canada*. AAAI Press, Menlo Park, CA, 87–93.

Tennenholtz, Moshe. 2000. Some tractable combinatorial auctions. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Austin*, *TX*. AAAI Press, Menlo Park, CA.

van Hoesel, Stan, Rudolf Müller. 2001. Optimization in electronic marketplaces: Examples from combinatorial auctions. *Netnomics* **3**(1) 23–33.

Walsh, William, Michael Wellman, Fredrik Ygge. 2000. Combinatorial auctions for supply chain formation. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*), *Minneapolis*, *MN*. ACM, New York, 260–269.

Wolsey, Laurence A. 1998. *Integer Programming*. John Wiley & Sons, New York.

Wurman, Peter R., Michael P. Wellman. 2000. AkBA: A progressive, anonymous-price combinatorial auction. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*), *Minneapolis*, *MN*. ACM, New York, 21–29.

Zinkevich, Martin, Avrim Blum, Tuomas Sandholm. 2003. On polynomial-time preference elicitation with value queries. *Proc. ACM Conf. Electronic Commerce (ACM-EC), San Diego, CA*. ACM, New York, 176–185.

Zurel, Edo, Noam Nisan. 2001. An efficient approximate allocation algorithm for combinatorial auctions. *Proc. ACM Conf. Electronic Commerce* (*ACM-EC*), *Tampa*, *FL*. ACM, New York, 125–136.