An Algorithm for Optimal Winner Determination in Combinatorial Auctions

Tuomas Sandholm

WUCS-99-01
January 28, 1999

An Algorithm for Optimal Winner Determination in Combinatorial Auctions*

Tuomas Sandholm
sandholm@cs.wustl.edu
Washington University
Department of Computer Science
One Brookings Drive, Campus Box 1045
St. Louis, MO 63130-4899

Abstract

Combinatorial auctions, i.e. auctions where bidders can bid on combinations of items, tend to lead to more efficient allocations than traditional auctions in multi-item auctions where the agents' valuations of the items are not additive. However, determining the winners so as to maximize revenue is $\mathcal{NP}$-complete. First, existing approaches for tackling this problem are reviewed: exhaustive enumeration, dynamic programming, approximation algorithms, and restricting the allowable combinations. Then we present our search algorithm for optimal winner determination. Experiments are shown on several bid distributions. The algorithm allows combinatorial auctions to scale up to significantly larger numbers of items and bids than prior approaches to optimal winner determination by capitalizing on the fact that the space of bids is necessarily sparsely populated in practice. The algorithm does this by provably sufficient selective generation of children in the search tree, by using a secondary search for fast child generation, by heuristics that are accurate and optimized for speed, and by four methods for preprocessing the search space.

*Patent pending. A highly optimized implementation of the algorithm is available for licensing both for research and commercial purposes. Please contact the author.

1 Introduction

Auctions are popular, distributed and autonomy preserving ways of allocating items among agents. They are relatively efficient both in terms of process and outcome. They are extensively used among human bidders and among software agents in a variety of task and resource allocation problems.[1] This paper focuses on auctions with multiple items to be allocated.

1.1 Sequential auction protocols

In a *sequential auction*, the items are auctioned one at a time. Determining the winners in such protocols is easy because that can be done by picking the highest bidder for each item separately. However, in many settings, bidders have preferences over bundles, i.e. combinations of items. This is often the case for example in electricity markets, equities trading, bandwidth auctions [McAfee and McMillan, 1996, McMillan, 1994], transportation exchanges [Sandholm, 1993, Sandholm, 1991, Sandholm, 1996a], pollution right auctions, auctions for airport landing slots [Rassenti *et al.*, 1982], and auctions for carrier-of-last-resort responsibilities for universal services [Kelly and Steinberg, 1998]. In such domains, bidding in sequential auction protocols is difficult. To determine her valuation for an item, the bidder needs to guess what items she will receive in later auctions. This requires speculation on what the others will bid in the future because that affects what items she will receive. Furthermore, what the others bid in the future depends on what they believe others will bid, etc. This counterspeculation introduces computational cost and other wasteful overhead. Moreover, in auctions with a reasonable number of items, such lookahead in the game tree is intractable, and then there is no known way to bid rationally. Bidding rationally would involve optimally trading off the cost of lookahead against the gains it provides, but that would again depend on how others strike that tradeoff. Furthermore, even if lookahead were computationally manageable, usually uncertainty remains about the others' bids because agents do not

[1] Auctions are usually discussed in the context of settings where the auctioneer wants to sell the items and get the highest possible payments for them while each bidder wants to acquire the items at the lowest possible price. However, settings in which the auctioneer wants to subcontract out tasks at the lowest possible prices and each bidder wants to handle the tasks at the highest possible payments are totally analogous.

have exact information about each other. This often leads to inefficient allocations where bidders fail to get the combinations they want and get ones they do not.

1.2 Parallel auction protocols

As an alternative to sequential auctions, a parallel auction design can be used. In a *parallel auction* the items are open for auction simultaneously and bidders may place their bids during a certain time period. This has the advantage that the others' bids partially signal to the bidder what the others' bids will end up being so the uncertainty and the need for lookahead is not as drastic as in a sequential auction.[2] However, the same problems prevail as in sequential auctions, albeit in a mitigated form.

In parallel auctions, an additional difficulty arises: each bidder would like to wait until the end to see what the going prices will be, and to optimize her bids so as to maximize payoff given the final prices. Because every bidder would want to wait, no bidding would commence. As a patch to this problem, activity rules have been used [McAfee and McMillan, 1996]. Each bidder has to bid at least a certain volume by predefined time points in the auction, otherwise the bidder's future rights are reduced in some prespecified manner. Unfortunately, the equilibrium bidding strategies in such auctions are not game-theoretically known. It follows that the outcomes of such auctions are unknown for rational bidders.

1.3 Methods for fixing inefficient allocations

In sequential and parallel auctions, the computational cost of lookahead and counterspeculation cannot be recovered, but one can attempt to fix the inefficient allocations that stem from the uncertainties discussed above.

One such approach is to set up an aftermarket where the bidders can exchange items among themselves after the auction has closed. While this approach can undo some inefficiencies, it may not lead to a Pareto efficient

[2] In sealed-bid implementations, the sequential and parallel variants are equivalent since neither reveals information during the auction process. Therefore, the advantages of parallel auctions only come into play in open-cry auctions where the bidders observe the others' bids.

allocation in general, and even if it does, that may take an impractically large number of exchanges among the agents [Sandholm, 1998].

Another approach is to allow bidders to retract their bids if they do not get the combinations that they want. For example, in the Federal Communications Committee's bandwidth auction the bidders were allowed to retract their bids [McAfee and McMillan, 1996]. In case of a retraction, the item was opened for reauction. If the new winning price was lower than the old one, the bidder that retracted the bid had to pay the difference. This guarantees that retractions do not decrease the auctioneer's payoff. However, it exposes the retracting bidder to considerable risk.

This risk can be eliminated by a *leveled commitment* protocol [Sandholm and Lesser, 1996, Sandholm and Lesser, 1995], where the decommitting penalties are set up front, possibly on a per item basis. This protocol allows the bidders to decommit but it also allows the auctioneer to decommit. A bidder may want to decommit for example if she did not get the combination that she wanted but only a subset of it. The auctioneer may want to decommit for example if he believes that he can get a higher price for the item later on. The leveled commitment protocol has interesting gaming aspects: the agents do not decommit truthfully because there is a chance that the other agent will decommit, in which case the former agent is freed from the contract obligations, does not have to pay the decommitment penalty, and will collect a penalty from the latter agent. We have shown that despite this gaming, in Nash equilibrium, the protocol can increase the expected payoff of both parties, and enable contracts which would not be individually rational to both parties via any full commitment contract [Sandholm and Lesser, 1996, Sandholm, 1996b]. Experimental results regarding leveled commitment contracts are presented for example in [Andersson and Sandholm, 1998a, Andersson and Sandholm, 1998b].

Yet another approach would be to sell options for decommitting, where the price of the option would be paid up front regardless of whether it is actually exercised.

Each one of the methods above can be used to implement bid retraction before and/or after the winning bids have been determined. While these methods can be used to try to fix inefficient allocations, it would clearly be desirable to get efficient allocations right away in the auction itself, so no fixing would be required. Combinatorial auctions hold significant promise toward that goal.

1.4 Combinatorial auction protocols

Combinatorial auctions can be used to overcome the need for lookahead and the inefficiencies that stem from the uncertainties [Rassenti *et al.*, 1982, Sandholm, 1993, McMillan, 1994, Sandholm, 1991]. In a combinatorial auction, bidders may place bids on combinations of items. This allows the bidders to express complementarities between items instead of having to speculate into an item's valuation the impact of possibly getting other, complementary items. For example, the Federal Communications Commission saw the desirability of combinatorial bidding in their bandwidth auctions, but it was not allowed due to perceived intractability of winner determination. This paper focuses on winner determination in combinatorial auctions where each bidder can bid on combinations of indivisible items, and any number of her bids can be accepted.

2 Winner determination in combinatorial auctions

The determination of winners—i.e. determining what items each bidder gets—is easy in non-combinatorial auctions. It can be done by picking the highest bidder for each item separately. This takes $O(am)$ time where a is the number of bidders, and m is the number of items.

Unfortunately, winner determination in combinatorial auctions is hard. Let M be the set of items to be auctioned, and let $m = |M|$. Then any agent, i, could place any bid, $b_i(S)$, for any combination $S \subseteq M$. The relevant bids are:

$$\bar{b}(S) = \max_{i \,\in\, \text{bidders}} b_i(S)$$

Let n be the number of these bids. Winner determination is the following problem, where the goal is to maximize the auctioneer's revenue:

$$\max_{\mathcal{X}} \sum_{S \in \mathcal{X}} \bar{b}(S)$$

where $\mathcal{X}$ is a valid outcome, i.e. an outcome where each item is allocated to only one bidder: $\mathcal{X} = \{S \subseteq M | S \cap S' = \emptyset \text{ for every } S, S' \in \mathcal{X}\}$.

2.1 Exhaustive enumeration

If each combination S has received at least one bid of positive price, the search space will look like Figure 1.

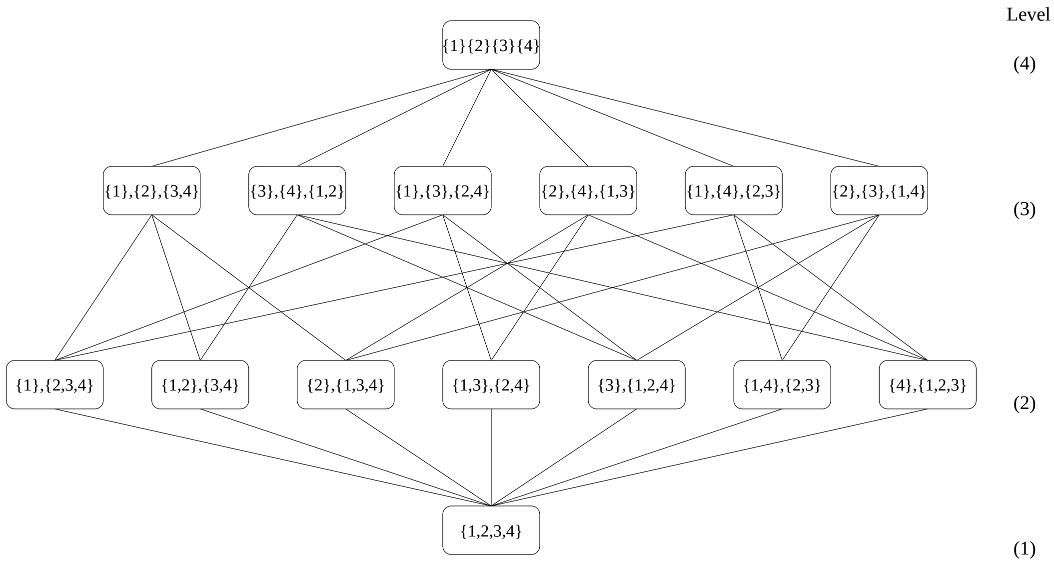

Figure 1: *Space of allocations in a 4-item example. Each node represents one possible allocation $\mathcal{X}$.*

The number of possible allocations, i.e. vertices in the graph, grows rapidly as the number of items increases. The exact number of allocations is

$$\sum_{q=1}^{m} Z(m,q) \tag{1}$$

where $Z(m,q)$ is the number of allocations with q accepted bids, i.e. the number of allocations on level q of the graph. The quantity $Z(m,q)$—also known as the Stirling number of the second kind—is captured by the following recurrence:

$$Z(m,q) = qZ(m-1,q) + Z(m-1,q-1), \tag{2}$$

where $Z(m,m) = Z(m,1) = 1$. This recurrence can be understood by considering the addition of a new item to a setting with $m-1$ items. The first term, $qZ(m-1,q)$, counts the number of allocations formed by adding the new item to one of the existing allocations. There are q choices because the existing allocations have q accepted bids. The second term, $Z(m-1, q-1)$, considers using the new item in a bid of its own, and therefore existing allocations with only $m-1$ previously accepted bids are counted.

The following proposition characterizes the asymptotic complexity in closed form. Proofs are given in the appendix.

Proposition 2.1 *The number of allocations is $O(m^m)$ and $\omega(m^{m/2})$.*

The number of allocations is so large that all allocations cannot be enumerated unless the number of items is extremely small—below a dozen or so in practice. Therefore, exhaustive enumeration is not a viable method for searching for the optimal allocation in most settings.

2.2 Dynamic programming

The allocations (Figure 1) can be searched more efficiently than exhaustive enumeration by dynamic programming [Rothkopf *et al.*, 1995]. Based on the $\bar{b}(S)$ function, the dynamic programming algorithm determines for each set S of items the highest possible revenue that can be acquired using only the items in S. The algorithm proceeds systematically from small sets to large ones. The needed optimal substructure property comes from the fact that for each set, S, the maximal revenue comes either from a single bid $\bar{b}(S)$, or from the sum of the maximal revenues of two disjoint exhaustive subsets of S. For each S, all possible subsets (together with that subset's complement in S) are tried.

The savings compared to exhaustive search come from the fact that the revenue maximizing solutions for the subsets need not be computed over and over again, but only once. The dynamic programming algorithm takes $O(3^m)$ and $\Omega(2^m)$ steps [Rothkopf *et al.*, 1995], which is a considerable saving over exhaustive enumeration, but still too complex to scale to large numbers of items—above about 25 in practice.

The dynamic programming algorithm executes the same steps independent of the number of bids. This is because the algorithm generates each combination S even if no bids have been placed on S. Interpreted positively this means that the auctioneer can determine *ex ante* how long winner determination will take regardless of the number of bids that will be received. Interpreted negatively this means that the algorithm will scale only to a small number of items even if the number of bids is relatively small. In Section 3 we present a search algorithm that avoids the generation of combinations for which bids have not been placed. That allows our algorithm to scale up to large numbers of items.

2.3 $\mathcal{NP}$-completeness

Some combinations of items may not have received any bids, so some of the allocations in the graph need not be considered. Thus the relevant question is not how many allocations there might be if the space of bids were completely populated, but instead, can the optimal allocation be found quickly, e.g. in polynomial time in the actual number of bids? Unfortunately no algorithm can find the optimal allocation in polynomial time in n, the number of bids submitted, unless $\mathcal{P} = \mathcal{NP}$:

Proposition 2.2 *Winner determination is $\mathcal{NP}$-complete.*

2.4 Polynomial time approximation algorithms

One way to attempt to achieve tractability is to try to find a reasonably good allocation $\mathcal{X}$ instead of an optimal one. One would then like to trade off the expected cost of additional computation (cost of the computational resources and cost associated with delaying the result) against the expected improvement in $\mathcal{X}$.

Instead of using expectations, one could try to devise an algorithm that will establish a worst case bound, i.e. guarantee that the value of the best allocation $\mathcal{X}$ is no more than some constant, k, times the value of the best $\mathcal{X}$ found by the algorithm. A considerable amount of research has focused on generating such approximation algorithms that execute in polynomial time. In the case of combinatorial auctions this means time that is polynomial in n, the number of bids received. The relevant approximation algorithms were developed for the weighted set packing problem or the weighted independent set problem, but they can be used for winner determination. In this section we translate the known algorithms and inapproximability results from the theory of combinatorial algorithms into the winner determination problem.

2.4.1 General case

Unfortunately, the following proposition shows that no polynomial time algorithm can be constructed for achieving a reasonable worst case guarantee in winner determination. The proposition follows from a recent inapproximability result of Håstad for a related problem [Håstad, 1999].

Proposition 2.3 (inapproximability) *No polynomial time algorithm can guarantee a bound $k \leq n^{1-\epsilon}$ for any $\epsilon > 0$ for the winner determination problem (unless $\mathcal{NP}$ equals probabilistic polynomial time).*

From a practical perspective the question of polynomial time approximation with worst case guarantees has been answered since algorithms that come very close to the inapproximability result have been constructed. The asymptotically best published polynomial time algorithm establishes a bound $k \in O(n(\log \log n / \log n)^2)$ [Halldórsson, 1995], and a slightly better algorithm is about to be published with a bound $k \in O(n/(\log n)^2)$ [Halldórsson, 1998b].

One could also ask whether randomized algorithms would help in the winner determination problem. It is conceivable that randomization could provide some improvement over the bounds that the two algorithms mentioned above provide. However, Proposition 2.3 applies to randomized algorithms as well, so no meaningful advantage could be gained from randomization.

Put together, the approach of constructing polynomial time approximation algorithms with worst case guarantees is a futile effort in the winner determination problem. Even a bound $k = 2$ would mean that the algorithm might only capture 50% of the available revenue, and usually $k \gg 2$ so the guarantee would be even weaker.

2.4.2 Special cases

While the general winner determination problem is inapproximable in polynomial time, one can do somewhat better in special cases where the bids have special structure. For example, there might be some cap on the number of items per bid, or there might be a cap on the number of bids with which a bid can share items.

The desired special structure could be enforced on the bidders by restricting the allowable bids. However, that can lead to the same inefficiencies as non-combinatorial auctions because bidders may not be allowed to bid on the combinations they want. Alternatively, the auctioneer can allow general bids and use these special case algorithms if the bids happen to exhibit the desired special structure.

This section reviews the best polynomial time algorithms for the known special cases. These algorithms were developed for the weighted independent set problem or the weighted set packing problem. Here we show how they apply to the winner determination problem:

1. If the bids have at most w items each, a bound $k = 2(w+1)/3$ can be established in $O(nw^2\Delta^w)$ time, where Δ is the number of bids that any given bid can share items with (note that $\Delta < n$) [Chandra and Halldórsson, 1999]. First, bids are greedily inserted into the solution on a highest-price-first basis. Then local search is used to improve the solution, and this search is terminated after a given number of steps. At each step, one new bid is inserted into the solution, and the old bids that share items with the new bid are removed from the solution. These improvements are chosen so as to maximize the ratio of the new bid's price divided by the sum of the prices of the old bids that would have to be removed.

2. Several algorithms have been developed for the case where each bid shares items with at most Δ other bids. A bound $k = \lceil(\Delta+1)/3\rceil$ can be established in linear time by partitioning the set of bids into $\lceil(\Delta+1)/3\rceil$ subsets such that $\Delta \leq 2$ in each subset, and then using dynamic programming to solve the weighted set packing problem in each subset [Halldórsson, 1998a]. Other polynomial time algorithms for this setting establish bounds $k = \Delta/2$, see [Hochbaum, 1983], and $k = (\Delta+2)/3$, see [Halldórsson and Lau, 1997].

3. A bound that depends only on the number of items, m, can also be established in polynomial time. An algorithm that establishes a bound $\sqrt{m}$ has recently been developed [Halldórsson, 1998b], but has not yet been published.

4. If the bids can be colored with c colors so that no two bids that share items have the same color, then a bound $k = c/2$ can be established in polynomial time [Hochbaum, 1983].

5. The bids have a κ-claw if there is some bid that shares items with κ other bids which themselves do not share items. If the bids are free of κ-claws, a bound $k = \kappa - 2 + \epsilon$ can be established with local search in $n^{O(1/\epsilon)}$ time [Halldórsson, 1998a]. Another algorithm establishes a bound $k = (4\kappa + 2)/5$ in $n^{O(\kappa)}$ time [Halldórsson, 1998a].

6. Let D be the largest d such that there is some subset of bids in which every bid shares items with at least d bids in that subset. Then, a bound $k = (D+1)/2$ can be established in polynomial time [Hochbaum, 1983].

Approximation algorithms for these known special cases have been improved repeatedly. There is also the possibility that probabilistic algorithms could improve upon the deterministic ones. In addition, it is possible that additional special cases with desirable approximability properties will be found. For example, while the current approximation algorithms are based on restrictions on the structure of bids and items, a new family of restrictions that will very likely lend itself to approximation stems from limitations on the prices. For example, if the function $\bar{b}(S)$ is close to additive, approximation should be easy. Unfortunately it does not seem reasonable for the auctioneer to restrict the bid prices or to eliminate outlier bids after the bids have been submitted. Setting an upper bound could reduce the auctioneer's revenue because higher bids would not occur. Setting a lower bound above zero would disable bidders with lower valuations from bidding, and if no bidder has a valuation above the bound, no bids on those combinations would be placed. That can again reduce the auctioneer's revenue. Although forcing such special structure does therefore not make sense, the auctioneer could capitalize on special price structure if such structure happens to be present in the bids without forcing.

Put together, considerable work has been done on approximation algorithms for special cases of combinatorial problems, and these algorithms can be used for special cases of the winner determination problem. However, these algorithms provide worst case guarantees that are so far from optimum that they are irrelevant for auctions in practice.

2.5 Restricting the combinations to guarantee optimal winner determination in polynomial time

If even more severe restrictions apply to the bids, winner determination can be carried out optimally in polynomial time. To capitalize on this idea in practice, the restrictions would have to be imposed by the auctioneer since they—at least the currently known ones [Rothkopf *et al.*, 1995]—are so severe that it is very unlikely that they would hold by chance. In this section we review the bid restrictions that have been suggested for achieving polynomial winner determination [Rothkopf *et al.*, 1995]:

1. If the bids have at most 2 items each, the winners can be optimally determined in $O(m^3)$ time using an algorithm for maximum-weight matching. The problem is $\mathcal{NP}$-complete already if the bids can have 3 items (this can be proven via a reduction from 3-set packing).

2. If the bids are of size 1 or of size greater than m/c, the winners can be optimally determined in $O(m(n_{large})^c)$ time, where n_{large} is the number of bids of size greater than m/c.

3. If the allowable combinations are in a tree structure, winners can be optimally determined in polynomial time. For example in Figure 2 left, a bid on 4 and 5 would be allowed while a bid for 5 and 6 would not. In tree structures the winners can be optimally determined by propagating information once up the tree. At every node, the best decision is to accept either an entirety bid for all the items in that node or the best allocations in the children of that node.

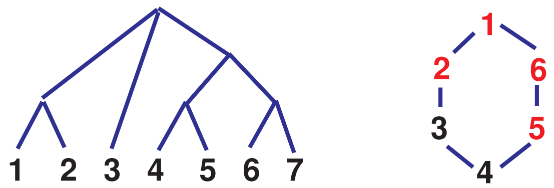

Figure 2: *Left: Allowable bids in a tree structure. Right: Interval bids.*

4. The items can be ordered and it can be required that bids are only placed on consecutive items. For example in Figure 2 right, a bid on 5,

6, 1, and 2 would be allowed while a bid on 5 and 1 would not. Without wrap-around, the winners can be optimally determined using dynamic programming. The algorithm starts from item 1, then does 1 and 2, then 1, 2, and 3, etc. The needed optimal substructure property comes from the fact that the highest revenue that can be achieved from items $1, 2, ..., h$ can be achieved either by picking a bid that has been placed on that entire combination, or by picking a bid that has been placed on the combination $g, ..., h$ and doing what was best for $1, ..., g-1$ (all choices $1 < g \leq h$ are tried). The algorithm runs in $O(m^2)$ time.

If wrap-around is allowed, the winners can be optimally determined in $O(m^3)$ time by rerunning the $O(m^2)$ algorithm m times, each time cutting the chain at a different point.

Imposing restrictions on the bids introduces some of the same inefficiencies that are present in non-combinatorial auctions because the bidders may be barred from bidding on the combinations that they want. There is an inherent tradeoff here between computational speed and economic efficiency. Imposing certain restrictions on bids achieves provably polynomial time winner determination but gives rise to economic inefficiencies.

3 Our optimal search algorithm

We recently generated another approach to optimal winner determination. It is based on highly optimized search. The motivation behind our approach is to

- allow bidding on all combinations unlike the approach of [Rothkopf *et al.*, 1995]. This is in order not to introduce any of the inefficiencies that occur in non-combinatorial auctions.
- strive for the optimal allocation. This maximizes the seller's revenue.
- completely avoid loops and redundant generation of vertices when searching the allocation graph (Figure 1).
- capitalize heavily on the sparseness of bids—unlike dynamic programming which uses the same amount of time irrespective of the number of bids. In practice the space of bids is necessarily extremely sparsely

populated. For example, even if there are only 100 items to be auctioned, there are $2^{100} - 1$ combinations, and it would take longer than the life of the universe to bid on all of them even if every person in the world submitted a bid per second. Sparseness of bids implies sparseness of the allocations $\mathcal{X}$ that need to be checked. Our algorithm constructively checks each allocation $\mathcal{X}$ that has positive value exactly once, and does not construct the other allocations. Therefore, unlike dynamic programming, our algorithm only generates those parts of the search space that are actually populated by bids. It follows that the disadvantage of our algorithm is that the run time depends on the number of bids received, while in dynamic programming it does not.

To achieve these goals, we use a search algorithm that generates a tree, see Figure 3. Each path in the tree consists of a sequence of disjoint bids, i.e. bids that do not share items. As a bid is added to the path, the bid price is added to the g-function. A path terminates when all items have been used on that path. At that point the path corresponds to a feasible allocation, and the revenue from that allocation, i.e. the g-value, can be compared to the best one found so far to determine whether the allocation is the best one so far. The best allocation found so far is stored, and once the search completes, that allocation is optimal.

The naive method of constructing the search tree would include all bids (that do not include items that are already on the path) as the children of each node. Instead, the following proposition enables a significant reduction of the branching factor by capitalizing on the fact that the order of the bids on a path does not matter.

Proposition 3.1 *Every allocation will be explored exactly once in the tree if the children of a node are those bids that*

- *include the item with the smallest index among the items that are not on the path yet, and*
- *do not include items that are already on the path.*

Our search algorithm restricts the children according to the proposition, see Figure 3. This can be seen for example at the first level because all the

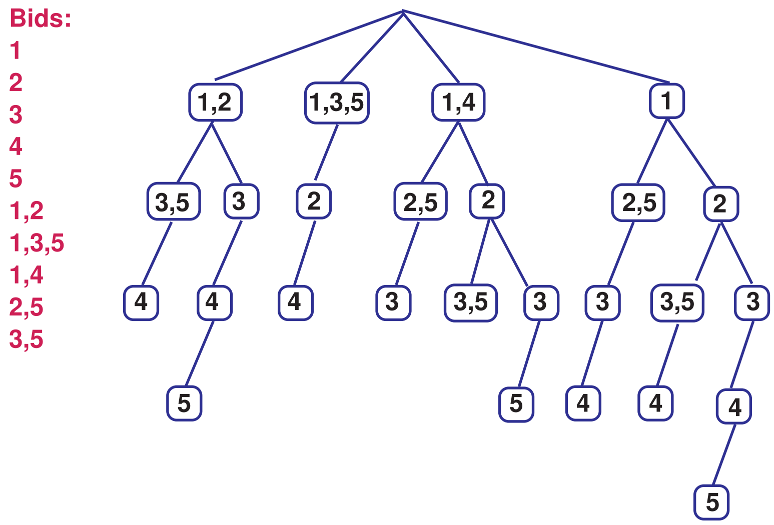

Figure 3: *A search tree generated by our algorithm.*

bids considered at the first level include item 1. The minimal index does not coincide with the depth of the search tree in general.

The auctioneer's revenue can increase if not every item has to be allocated, i.e. he can keep items. That can be profitable if some item has received no bids on its own. For example, say there is no bid for item 1, a \$5 bid for item 2, and a \$3 bid for the combination of 1 and 2. Then it is more profitable for the auctioneer to keep 1 and to allocate 2 alone than it would be to allocate both 1 and 2 together. Such optimization can be implemented by placing dummy bids of price zero on those individual items that received no bids alone, see Figure 3. For example, if item 1 had no bids on it alone and dummies were not used, the tree under 1 would not be explored and optimality could be lost. When dummy bids are used, the resulting search generates each allocation that has positive revenue exactly once—and searches through no other allocations. This guarantees that the algorithm finds the optimal solution. Throughout the rest of the paper, we use this dummy bid technique.

3.1 Optimized generation of children

The main search algorithm uses a secondary depth-first-search (DFS) to quickly determine the children of a node. The secondary search occurs in a data structure which we call the *Bidtree*. It is a binary tree in which the bids are inserted up front as the leaves. Only those parts of the tree are generated for which bids are received, see Figure 4. What makes the data

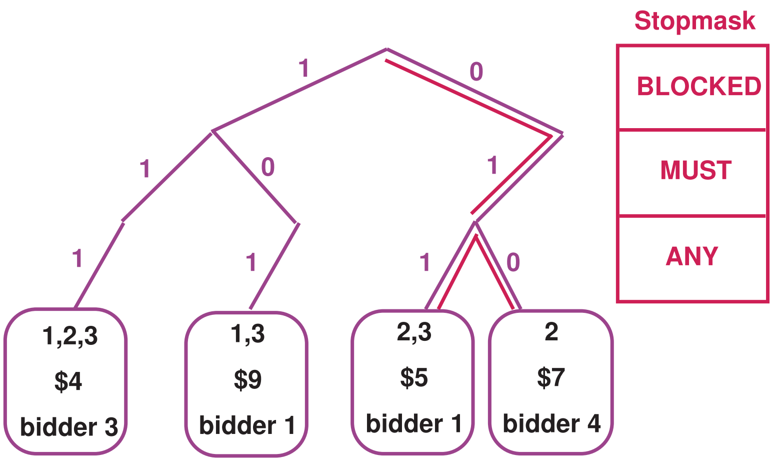

Figure 4: *The Bidtree data structure.*

structure special is the use of a *Stopmask*. The Stopmask is a vector with one variable for each auctioned item. If the variable corresponding to an item has the value BLOCKED, those parts of the Bidtree are pruned instantaneously, and in place, that contain bids containing that item. In other words, search in the Bidtree will never progress left at that level. If the item's variable has the value MUST, all other parts of the Bidtree are pruned instantaneously and in place, i.e. search cannot progress right at that level. The value ANY corresponds to no pruning based on that item: the search may go left or right.

To start, the first item has value MUST in the Stopmask, and the others have ANY. The first child of any given node in the main search is determined by a DFS from the top of the Bidtree. The siblings of that child are determined by continuing that DFS by backtracking in the Bidtree after the main search has explored the tree under the first child. As a bid is appended to the path of the main search, BLOCKED is inserted in the Stopmask for each item of that bid. That implements the branching reduction of the main search based on the second bullet of Proposition 3.1. MUST is inserted at the unallocated item with the smallest index. That implements the branching reduction of the main search based on the first bullet of Proposition 3.1. These MUST and BLOCKED values are changed back to ANY when backtracking a bid from the path of the main search, and MUST is reallocated to the place where it was before that bid was appended to the path. Note that the secondary DFS never needs to backtrack above the variable that contains MUST because all items with smaller indices than that are already used on the path of the main search.

The secondary search can be implemented to execute in place, i.e. without memory allocation during search. That is accomplished via the observation that recursion or an open list is not required because in DFS, to decide where to go next, it suffices to know where the search focus is now, and from where it most recently came.

3.2 Anytime winner determination via depth-first-search (DFS)

We first implemented the main search as DFS which executes in linear space. The depth-first strategy causes feasible allocations to be found quickly (the first one is generated in linear time when the first search path ends), and the solution improves monotonically since the algorithm keeps track of the best solution found so far. This implements the anytime feature: if the algorithm does not complete in the desired amount of time, it can be terminated prematurely, and it guarantees a feasible solution that improves monotonically. When testing the anytime feature, it turned out that in practice most of the revenue was generated early on as desired, and there were diminishing returns to computation.

3.3 Preprocessing

Our algorithm preprocesses the bids in four ways to make the main search faster without compromising optimality. The preprocessors could also be used in conjunction with other approaches to winner determination than our search algorithm. The next subsections present the preprocessors in the order in which they are executed.

3.3.1 PRE1: Keep only the highest bid for a combination

As a bid arrives, it is inserted into the Bidtree. If a bid for the same S already exists in the Bidtree, only the bid with the higher price is kept, and the other bid is discarded. We break ties in favor of keeping the earlier bid.

3.3.2 PRE2: Remove provably noncompetitive bids

This preprocessor removes bids that are provably noncompetitive. A bid (*prunee*) is noncompetitive if there is some disjoint collection of subsets of that bid such that the sum of the bid prices of the subsets exceeds or equals the price of the prunee bid. For example, a \$10 bid for items 1, 2, 3, and 4 would be pruned by a \$4 bid for items 1 and 3, and a \$7 bid for items 2 and 4.

To determine this we search, for each bid (potential prunee), through all combinations of its disjoint subset bids. This is the same DFS as the main search except that it restricts the search to those bids that only include items that the prunee includes (Figure 5): BLOCKED is kept in the Stopmask for other items. This ensures that only subset bids contribute to the pruning.

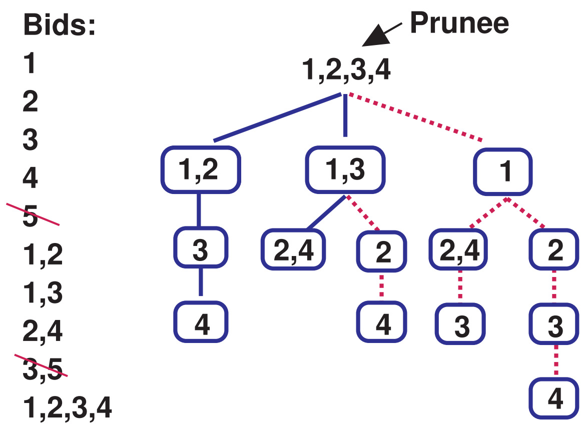

Figure 5: *A search tree generated for one prunee in PRE2. The dotted paths are not generated because pruning occurs before they are reached.*

Especially with bids that contain a large number of items, PRE2 can take more time than it saves in the main search. In the extreme, if some bid contains all items, the preprocessing search with that bid as the prunee is the same as the main search (except for one main search path that contains that bid only). To save preprocessing time, PRE2 is carried out partially. With such partial pruning, some of the noncompetitive bids are left unpruned. That will not compromise optimality of the main search although the main search can become slower. We implemented two ways of restricting PRE2:

1. A cap, Γ, on the number of pruner bids that can be combined to try to prune a particular prunee bid. This limits the depth of the search in PRE2 to Γ.

2. A cap, Φ, on the number of items in a prunee bid. Longer bids would then not be targets of pruning. This entails a cap, Φ, on tree depth. It also tends to exclude wide trees because long prunees usually lead to trees with large branching factors.

With either method, PRE2 takes $O(nn^{cap}m)$ time, which is polynomial for a constant cap (there are n prunees, the tree for each is $O(n^{cap})$, and finding a child in the Bidtree is $O(m)$). The latter method is usually preferable. It does not waste computation on long prunees which take a lot of preprocessing time and do not significantly increase the main search time. This is because the main search is shallow along the branches that include long bids due to the fact that each item can occur only once on a path and a long bid uses up many items. Second, if the bid prices are close to additive, the former method does not lead to pruning when a path is cut prematurely based on the cap.

3.3.3 PRE3: Decompose bids into connected sets

The bids are partitioned into sets such that no item is shared by bids from different sets. PRE4 and the main search are then done in each set of bids independently, and using only items included in the bids of the set. The sets are determined as follows. We define a graph where bids are vertices, and two vertices share an edge if the bids share items. We generate an adjacency list representation of the graph in $O(mn^2)$ time. We use DFS to generate a depth-first forest of the graph in $O(n+m)$ time. Each tree in the forest is then a set with the desired property.

3.3.4 PRE4: Mark noncompetitive tuples of bids

Noncompetitive tuples of disjoint bids are marked so that they need not be considered on the same path in the main search. For example, the pair of bids \$5 for items 1 and 3, and \$4 for items 2 and 5 is noncompetitive if there is a bid of \$3 for items 1 and 2, and a bid of \$7 for items 3 and 5. Noncompetitive tuples are determined as in PRE2 except that now each prunee is a virtual bid that contains the items of the bids in the tuple, and the prunee price is the sum of the prices of those bids.

For computational speed, we only mark 2-tuples, i.e. pairs of bids. A pair of bids is excluded also if the bids share items. PRE4 is used as a partial preprocessor like PRE2, with caps Γ' or Φ' instead of Γ or Φ. PRE4 runs in $O(n^2 n^{cap} m)$ time. Handling 3-tuples would increase this to $O(n^3 n^{cap} m)$, etc. Handling large tuples also slows the main search because it needs to ensure that noncompetitive tuples do not exist on the path.

As a bid is appended to the path, it excludes from the rest of the path those other bids that constitute a noncompetitive pair with it. Our algorithm determines this quickly as follows. For each bid, a list of bids to exclude is determined in PRE4. In the main search, an exclusion count is kept for each bid, starting at 0. As a bid is appended to the path, the exclusion counts of those bids that it excludes are incremented. As a bid is backtracked from the path, those exclusion counts are decremented. Then, when searching for bids to append to the main search path from the Bidtree, only bids with exclusion count 0 are accepted.[3]

3.4 IDA* and heuristics

We sped up the main search by using an iterative deepening A* (IDA*) search strategy [Korf, 1985] instead of DFS. The search tree, use of the Bidtree, and the preprocessors stay the same. In practice, IDA* finds the provably optimal allocation well before the entire search tree (Figure 3) has been traversed. At

[3]PRE2 and PRE4 could be converted into anytime preprocessors without compromising optimality by starting with a small cap, conducting the searches, increasing the cap, reconducting the searches, etc. Preprocessing would stop when it is complete (cap $= n$), the user decides to stop it, or some other stopping criterion is met. PRE2 and PRE4 could also be converted into approximate preprocessors by allowing pruning when the sum of the pruners' prices exceeds a fixed fraction of the prunee's price. This would allow more bids to be pruned which can make the main search faster, but it can compromise optimality.

each iteration of IDA*—except the last—the IDA* threshold gives an upper bound on solution quality. It can be used, for example, to communicate search progress to the auctioneer.

Since winner determination is a maximization problem, the heuristic function h should never underestimate the revenue from the items that are not yet allocated in bids on the path because that could lose optimality. We designed two heuristics that never underestimate:

1. $$h = \sum_{i \,\in\, \text{unallocated items}} c(i) \quad \text{where} \quad c(i) = \max_{S|i \in S} \frac{\bar{b}(S)}{|S|}$$

2. As above, but accuracy is increased by recomputing $c(i)$ every time a bid is appended to the path since some combinations S are excluded: some of their items are on the path, or they constitute a noncompetitive pair with some bid on the path.

We use (2) with several methods for speeding it up. A tally of h is kept, and only some of the $c(i)$ values in h need to be updated when a bid is appended to the path. In PRE4 we precompute for each bid the list of items that must be updated: items included in the bid and in bids that are on the bid's exclude list. To make the update even faster, we keep a list for each item of the bids in which it belongs. The $c(i)$ value is computed by traversing that list and choosing the highest $\frac{\bar{b}(S)}{|S|}$ among the bids that have exclusion count 0. So, recomputing h takes $O(\underline{m}\underline{n})$ time, where $\underline{m}$ is the number of items that need to be updated, and $\underline{n}$ is the (average or greatest) number of bids in which those items belong.[4]

On the last IDA* iteration, the IDA* threshold is always incremented to equal the revenue of the best solution found so far in order to avoid futile search. In other words, once the first solution is found, the algorithm converts to branch-and-bound with the same heuristic.

[4]PRE2 and PRE4 use DFS because due to the caps their execution time is negligible compared to the main search time. Alternatively they could use IDA*. Unlike in the main search, the $c(i)$ values should be computed using only combinations S that are subsets of the prunee. The threshold for IDA* can be set to equal the prunee bid's price (or a fraction thereof in the case of approximation), so IDA* will complete in one iteration. Finally, care needs to be taken that the heuristic and the tuple exclusion are handled correctly since they are based on the results of the preprocessing itself.

4 Experimental setup

Not surprisingly, the worst case complexity of the main search is exponential in the number of bids. However, unlike dynamic programming, this is complexity in the number of bids actually received, not in the number of allowable bids. To determine the efficiency of the algorithm in practice, we ran experiments on a general-purpose uniprocessor workstation (360MHz Sun Ultra 60 with 256 MRAM) in C++ with four different bid distributions:

- **Random:** For each bid, pick the number of items randomly from $1, 2, ..., m$. Randomly choose that many items without replacement. Pick the price randomly 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.

If the same bid was generated twice, the new version was deleted and regenerated. So if the generator was asked to produce for example 500 bids, it produced 500 different bids.

We let all the bids have the same bidder. This conservative method causes PRE1 to prune no bids. In practice, the chance that two agents bid on the same combination of items is often small anyway because the number of combinations is large ($2^m - 1$). However, in some cases PRE1 is very effective. In the extreme, it prunes all of the bids except one if all bids are placed on the same combination by different bidders.

5 Experimental results

We focus on IDA* because it was two orders of magnitude faster than DFS. We lower the IDA* threshold between iterations to 95% of the previous threshold or to the highest $f = g + h$ that subceeded the previous threshold, whichever is smaller. Experimentally, this tended to be a good rate of decreasing the threshold. If it is decreased too fast, the overall number of search nodes increases because the last iteration becomes large. If it is decreased too slowly, the overall number of search nodes increases because new iterations repeat a large portion of the search from previous iterations.

For PRE2, the cap $\Phi = 30$ gave a good compromise between preprocessing time and main search time. For PRE4, $\Phi' = 20$ led to a good compromise. These values are used in the rest of the experiments. With these caps, the hard problem instances with short bids get preprocessed completely, and PRE2 and PRE4 take negligible time compared to the main search because the trees under such short prunees are small. The caps only take effect in the easy cases with long bids. In the uniform distribution all bids are the same length, so PRE2 does not prune any bids because no bid is a subset of another.

As expected, PRE3 saved significant time on the uniform and decay distributions by partitioning the bids into sets when the number of bids was small compared to the number of items, and the bids were short. In almost all experiments with random and weighted random, all bids fell in the same set because the bids were long. In real world combinatorial auctions it is likely that the number of bids will significantly exceed the number of items which would suggest that PRE3 does not help. However, most bids will usually be short, and the bidders' interests often have special structure which leads to some items being independent of each other, and PRE3 will automatically capitalize on that.

The main search generated 35,000 nodes per second when the number of items was small, e.g. 25, and the bids were short. This rate decreased slightly with the number of bids, but significantly with the number of items and bid size. With the random distribution with 400 items and 2000 bids, the search generated only 9 nodes per second. However, the algorithm solved these cases easily because the search paths were short and the heuristic focused the search well. Long bids make the heuristic and exclusion checking slower but the search tree shallower which makes them easier for our algorithm than short bids overall. This observation is further supported by the results below. Each point in each graph represents an average over 20 problem instances. The search times presented include all preprocessing times.

The random distribution was easy since the search was shallow because the bids were long, see Figure 6 left. The weighted random distribution was even easier, see Figure 6 right. The curves become closer together on the logarithmic value axis as the number of items increases, which means that search time is polynomial in items. In the weighted random case, the curves are sublinear meaning that search time is polynomial in bids as well, while in the unweighted case they are roughly linear meaning that search time is exponential in bids.

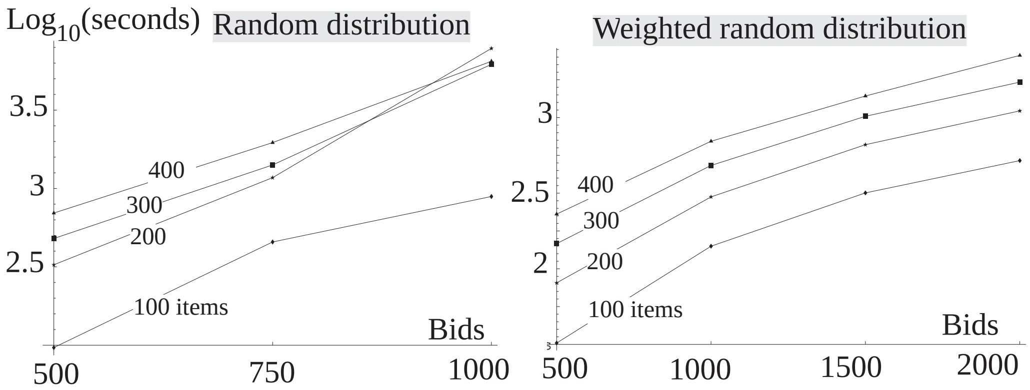

Figure 6: *Search time for the random and weighted random distributions. In the random distribution, the point with 1,000 bids and 200 items is unusually high due to one hard outlier among the 20 problem instances.*

The uniform distribution was harder, see Figure 7 left. The bids were shorter so the search was deeper. The curves are roughly linear so complexity is exponential in bids. The spacing of the curves does not decrease significantly indicating that complexity is exponential in items as well. Figure 7 right shows the complexity decrease as the bids get longer, i.e. the search gets shallower.

The decay distribution was also hard, see Figure 8 left. However, the curves become closer together on the logarithmic value axis as the number of items increases, which means that complexity is polynomial in items. Complexity first increases in α, and then decreases, Figure 8 right. Left of the maximum, PRE3 decomposes the problem leading to small, fast searches. The hardness peak moves left as the number of bids grows because the decomposition becomes less successful. Right of the maximum, all bids are in

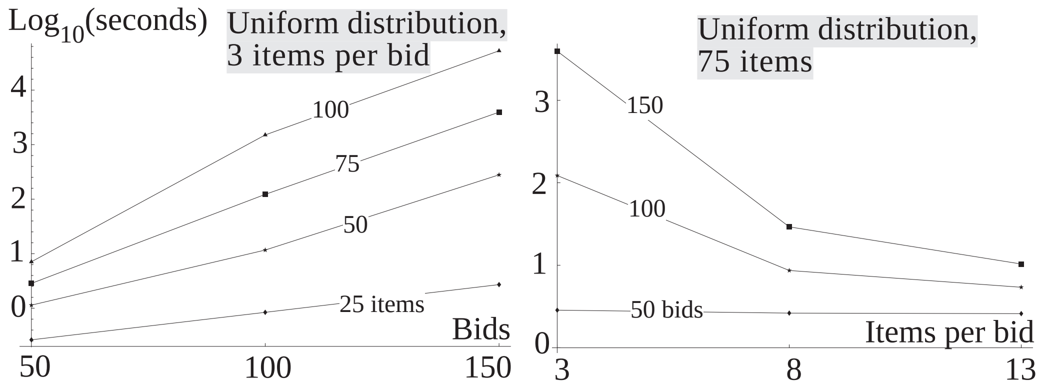

Figure 7: *Search time for the uniform distribution.*

the same set. The complexity then decreases with α because longer bids lead to shallower search.

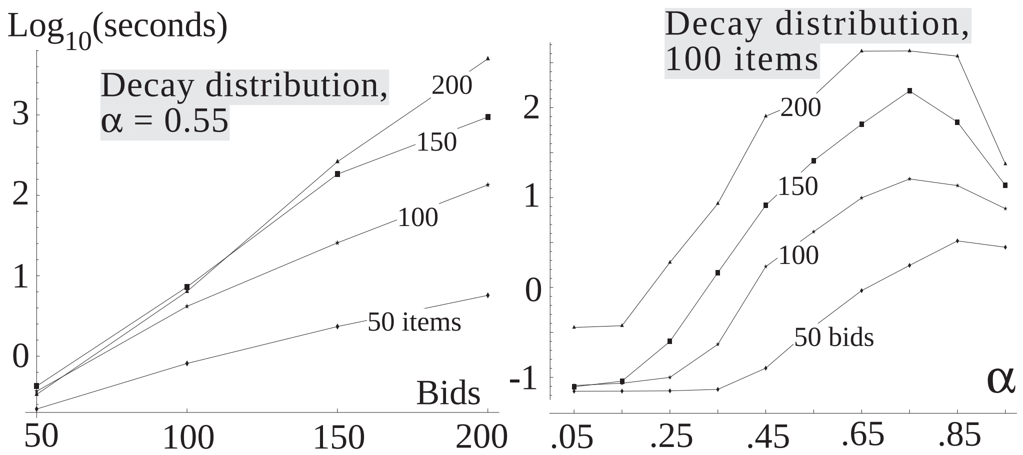

Figure 8: *Search time for the decay distribution.*

6 Other applications for the algorithms

The algorithms for winner determination can also be used to solve weighted set packing problems, weighted independent set problems, and weighted maximum clique problems because they are in fact the same problem. In addition they can be used for coalition structure generation in characteristic function games. The characteristic function assigns a value to each potential coalition of agents. The objective is to partition the agents exhaustively into disjoint coalitions so as to maximize the sum of the values of those coalitions. The agents correspond to items, coalitions to combinations, and coalition values to bid prices. This problem is analogous to the winner determination problem if the coalitions have nonnegative values. This can usually be achieved by normalizing if it does not hold up front. However, coalition structure generation differs from winner determination in two major ways.

First, the values of most coalitions are usually nonzero so the space of "bids" is densely populated. That reduces the usefulness of algorithms that capitalize on the sparseness of bids. This suggests the use of the dynamic programming algorithm, but that will not scale up past small numbers of agents here either.

Second, in coalition structure generation the values of coalitions might not be known to any party, and only values of complete coalition structures might be known, or they become known during a search process. [Sandholm *et al.*, 1998] recently devised an approximation algorithm with worst case guarantees for this case which is harder than the case where the values of coalitions are observed.

7 Conclusions and future research

Combinatorial auctions, i.e. auctions where bidders can bid on combinations of items, tend to lead to more efficient allocations than traditional auctions in multi-item auctions where the agents' valuations of the items are not additive. This is because the users can express complementarities in their bids, and the winner determination algorithm will take these into account. This paper tackled winner determination in combinatorial auctions where each bidder can bid on unrestricted combinations of items, and any number of her bids can get accepted. In such settings, determining the winners so as to maximize the auctioneer's revenue is $\mathcal{NP}$-complete.

The space of allocations was first explicated, and closed form asymptotic bounds on the number of allocations were derived. The number of allocations is so large that exhaustive enumeration will only work with a very small number of items. Dynamic programming avoids some of the redundancy of exhaustive enumeration, but it does not scale beyond auctions with a small number of items because it generates the entire search space independent of what bids have actually been received.

The approach of compromising optimality to achieve polynomial time winner determination is futile if one is interested in worst case approximation guarantees: a strong inapproximability result applies. If the combinations are restricted, somewhat better guarantees can be established by known approximation algorithms for the weighted independent set problem and the weighted set packing problem, but the guarantees remain so weak that they are irrelevant in the domain of auctions in practice. The main hope for practical approximations for special cases lies in new forms of special structure—especially on bid prices—and possibly in better algorithms such as probabilistic ones.

By imposing severe restrictions on the allowable combinations, optimal winner determination can be guaranteed in polynomial time. However, these restrictions introduce some of the same economic inefficiencies that are present in non-combinatorial auctions.

To tackle the limitations of the existing approaches to winner determination, we developed a search algorithm for optimal winner determination. Via experiments on several different bid distributions we showed that it significantly enlarges the envelope of input sizes for which combinatorial auctions with optimal winner determination are computationally feasible. The highly optimized algorithm achieves this mainly by capitalizing on the fact that the space of bids is necessarily sparsely populated in practice. Unlike dynamic programming, it generates only the populated parts of the search space. The algorithmic methods to implement this include provably sufficient selective generation of children in the search tree, a secondary search to find children quickly, and heuristics that are relatively accurate and optimized for speed. The algorithm also preprocesses the search space by keeping only the highest bid for each combination, by removing bids that are provably noncompetitive (this is determined via search), by decomposing the problem into independent parts, and by marking noncompetitive tuples of bids (this is again determined via search). We believe that our algorithm will make the difference between being able to use a combinatorial auction design in many practical markets and not.

The IDA* search algorithm, used in our main search, can easily be distributed across multiple computers for additional speed, see e.g. [Reinefeld and Schnecke, 1994]. The burden of search could also be imposed on the bidders by giving each bidder a portion of the search space to explore. This introduces two risks. First, a bidder may only search her portion partially so as to save computational effort. Such free-riding can be desirable to her since the other bidders are likely to find very good allocations anyway—assuming that they themselves do not free-ride. Second, a bidder may report a suboptimal allocation if that allocation leads to higher payoff for her. To prevent these problems, the auctioneer can randomly select one or more bidders after they have reported the best solutions that they found, and re-search their portions. If, in some portion, the auctioneer finds a better solution than the reported one, the bidder gets caught of fraudulent searching, and a penalty can be imposed. If the penalty is high enough compared to the cost of computation and compared to the maximal gain from reporting insincerely, and if the probability of getting checked upon is sufficiently high, each bidder is motivated to search her portion truthfully.

The algorithm can also be used to solve weighted set packing, weighted independent set, and weighted maximum clique problems because they are in fact the same problem. So is coalition structure generation in characteristic function games.

Both our algorithm, and the existing methods for winner determination that were reviewed, are based on the common implicit assumption that bids are superadditive: $\bar{b}(S \cup S') \geq \bar{b}(S) + \bar{b}(S')$. But what happens if agent 1 bids $b_1(\{1\}) = \$5$, $b_1(\{2\}) = \$4$, and $b_1(\{1,2\}) = \$7$, and there are no other bidders? The auctioneer could allocate items 1 and 2 to agent 1 separately, and that agent's bid for the combination would value at $\$5 + \$4 = \$9$ instead of \$7. So, the current techniques focus on capturing synergies (positive complementarities) among items. In practice, local subadditivities can occur as well. As a simple example, when bidding for a landing slot for a plane, the bidder is willing to take any one of a host of slots, but does not want more than one. We address this issue in our Internet auction house prototype which is part of our electronic commerce server called *eMediator*. We developed a protocol where the bidders can submit *XOR-bids*, i.e. bids on combinations such that only one of the combinations can get accepted. This allows the bidders to express general preferences with both positive and negative complementarities, see also [Rassenti *et al.*, 1982]. We also allow bids to state multiple units of each item, and we allow the bidders to submit multiple mutually nonexclusive XOR-bids. For these more general settings, the winner determination algorithms in our auction server are fast but they do not guarantee optimality of the allocation. Our current work focuses on developing optimal algorithms for these settings. The negative results, $\mathcal{NP}$-completeness and inapproximability, apply to these setting as well. Finally, we are developing winner determination algorithms for combinatorial double auctions which include multiple buyers and multiple sellers.

8 Acknowledgment

I thank Kate Larson who significantly contributed to the proof of Proposition 2.1.

References

[Andersson and Sandholm, 1998a] Martin R Andersson and Tuomas W Sandholm. Leveled commitment contracting among myopic individually rational agents. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS)*, pages 26–33, Paris, France, July 1998.

[Andersson and Sandholm, 1998b] Martin R Andersson and Tuomas W Sandholm. Leveled commitment contracts with myopic and strategic agents. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 38–45, Madison, WI, July 1998.

[Chandra and Halldórsson, 1999] Barun Chandra and Magnús M. Halldórsson. Greedy local search and weighted set packing approximation. In *10th Annual SIAM-ACM Symposium on Discrete Algorithms (SODA)*, January 1999. To appear.

[Comtet, 1974] L Comtet. *Advanced Combinatorics*. D. Reidel Pub. Co., 1974.

[Cormen *et al.*, 1990] Thomas H Cormen, Charles E Leiserson, and Ronald L Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[Halldórsson and Lau, 1997] Magnús M. Halldórsson and H C Lau. Low-degree graph partitioning via local search with applications to constraint satisfaction, max cut, and 3-coloring. *Journal of Graph Algo. Applic.*, 1(3):1–13, 1997.

[Halldórsson, 1995] Magnús M. Halldórsson. Approximations via partitioning. Technical Report IS-RR-95-0003F, School of Information Science, Japan Advanced Institute of Science and Technology, Hokuriku, Japan, 1995.

[Halldórsson, 1998a] Magnús M. Halldórsson. Approximations of independent sets in graphs. In K. Jansen and J. Rolim, editors, *The First International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 1–14, Aalborg, Denmark, July 1998. Springer LNCS 1444.

[Halldórsson, 1998b] Magnús M. Halldórsson. Personal communications. 1998. September 24th.

[Håstad, 1999] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 1999. To appear. Draft: Royal Institute of Technology, Sweden, 8/11/98. Early version: Proc. 37th IEEE Symposium on Foundations of Computer Science (1996), 627—636.

[Hochbaum, 1983] Dorit S. Hochbaum. Efficient bounds for the stable set, vertex cover, and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.

[Karp, 1972] R M Karp. Reducibility among combinatorial problems. In Raymond E Miller and James W Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, NY, 1972.

[Kelly and Steinberg, 1998] Frank Kelly and Richard Steinberg. A combinatorial auction with multiple winners for universal services. Technical report, University of Cambridge, June 1998.

[Korf, 1985] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[McAfee and McMillan, 1996] R Preston McAfee and John McMillan. Analyzing the airwaves auction. *Journal of Economic Perspectives*, 10(1):159–175, 1996.

[McMillan, 1994] John McMillan. Selling spectrum rights. *Journal of Economic Perspectives*, 8(3):145–162, 1994.

[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, 1982.

[Reinefeld and Schnecke, 1994] Alexander Reinefeld and Volker Schnecke. AIDA* - asynchronous parallel IDA*. In *10th Canadian Conf. on AI*, pages 295–302, Banff, Canada, 1994.

[Rothkopf *et al.*, 1995] Michael H Rothkopf, Aleksandar Pekeč, and Ronald M Harstad. Computationally manageable combinatorial auctions. Technical Report RRR 13-95, Rutgers Center for Operations Research, 1995. To appear in Management Science.

[Sandholm and Lesser, 1995] Tuomas W Sandholm and Victor R Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, pages 328–335, San Francisco, CA, June 1995. Reprinted in *Readings in Agents*, Huhns and Singh, eds., pp. 66–73, 1997.

[Sandholm and Lesser, 1996] Tuomas W Sandholm and Victor R Lesser. Advantages of a leveled commitment contracting protocol. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 126–133, Portland, OR, August 1996.

[Sandholm *et al.*, 1998] Tuomas W Sandholm, Kate S Larson, Martin R Andersson, Onn Shehory, and Fernando Tohmé. Anytime coalition structure generation with worst case guarantees. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 46–53, Madison, WI, July 1998.

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

[Sandholm, 1993] Tuomas W Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 256–262, Washington, D.C., July 1993.

[Sandholm, 1996a] Tuomas W Sandholm. Limitations of the Vickrey auction in computational multiagent systems. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS)*, pages 299–306, Keihanna Plaza, Kyoto, Japan, December 1996.

[Sandholm, 1996b] Tuomas W Sandholm. *Negotiation among Self-Interested Computationally Limited Agents*. PhD thesis, University of Massachusetts, Amherst, 1996. Available at http:// www.cs.wustl.edu/ ~sandholm/ dissertation.ps.

[Sandholm, 1998] Tuomas W. Sandholm. Contract types for satisficing task allocation: I theoretical results. In *AAAI Spring Symposium Series: Satisficing Models*, pages 68–75, Stanford University, CA, March 1998.

A Proofs

Proof. (Proposition 2.1). We first prove that the number of allocations is $O(m^m)$. Let there be m items. Let there be a set of *locations* where combinations can form, at most one combination per location. Let the number of locations be m. This allows for any allocation to form since an allocation can have at most m combinations. Now, say that the items get placed in locations one item at a time. Each item could be placed in m different locations, and there are m items to place. Therefore, the number of possible overall assignments is m^m. Thus, the number of allocations is $O(m^m)$.[5]

[5]Note that m^m overestimates the number of allocations. Some allocations are counted multiple times because for any given combination in the allocation, the allocation is counted once for each location where the combination can form (although it should be counted only once).

What remains to be proven is that the number of allocations is $\omega(m^{m/2})$. We use the following lemma in the proof.

Lemma A.1 *Let a and b be positive integers. Then $\lceil \frac{a}{b} \rceil! \leq (\frac{a}{b})^{\frac{a}{b}}$ for $\frac{a}{b} \geq 2$.*

Proof.

$$\begin{aligned}
\frac{\lceil \frac{a}{b} \rceil!}{(\frac{a}{b})^{\frac{a}{b}}} \leq \frac{\frac{a}{b}+1}{\frac{a}{b}} \cdot \frac{\frac{a}{b}}{\frac{a}{b}} \cdot \frac{\frac{a}{b}-1}{\frac{a}{b}} \cdots \frac{2}{\frac{a}{b}} &\leq \frac{\frac{a}{b}+1}{\frac{a}{b}} \cdot \frac{\frac{a}{b}-1}{\frac{a}{b}} \\
&= (1+\frac{b}{a})(1-\frac{b}{a}) \\
&= 1-(\frac{b}{a})^2 \\
&\leq 1
\end{aligned}$$

Therefore $\lceil \frac{a}{b} \rceil! \leq (\frac{a}{b})^{\frac{a}{b}}$. □

One way to count the number of allocations is to use *Bell numbers*. The Bell number B_m is equal to the number of ways a set of m elements can be partitioned into nonempty subsets. That is, B_m is equal to the number of allocations that can be generated with m items. There are several ways of calculating Bell numbers, including Dobinski's formula [Comtet, 1974]:

$$B_m = \frac{1}{e} \sum_{i=0}^{\infty} \frac{i^m}{i!}$$

To show that the number of allocations is $\omega(m^{m/2})$ it suffices [Cormen *et al.*, 1990] to show that

$$\lim_{m \to \infty} \frac{B_m}{m^{m/2}} = \infty$$

Since each term in the series of Dobinski's formula is positive, it suffices to take one term, $\frac{i^m}{i!}$ and show that

$$\lim_{m \to \infty} \frac{\frac{i^m}{i!}}{m^{m/2}} = \infty$$

Set $i = \lceil \frac{m}{b} \rceil$ for some constant b, where $b > 2$. Then the expression we are interested in is

$$\frac{(\lceil \frac{m}{b} \rceil)^m}{(\lceil \frac{m}{b} \rceil)! m^{m/2}} \geq \frac{(\frac{m}{b})^m}{(\lceil \frac{m}{b} \rceil)! m^{m/2}}$$

$$
\begin{aligned}
&\geq \frac{(\frac{m}{b})^m}{(\frac{m}{b})^{m/b} m^{m/2}} \\
&= \frac{m^{\frac{m(b-2)}{2b}}}{b^{\frac{m(b-1)}{b}}}
\end{aligned}
$$

We now calculate

$$
\lim_{m \to \infty} \frac{m^{\frac{m(b-2)}{2b}}}{b^{\frac{m(b-1)}{b}}}
$$

Since the natural logarithm and exponential functions are continuous, we can calculate the limit as follows. Let

$$
\begin{aligned}
\phi(m) &= \ln \frac{m^{\frac{m(b-2)}{2b}}}{b^{\frac{m(b-1)}{b}}} \\
&= \frac{m(b-2)}{2b} \ln m - \frac{m(b-1)}{b} \ln b \\
&= \frac{m}{b}[\frac{b-2}{2} \ln m - (b-1) \ln b]
\end{aligned}
$$

Then

$$
\begin{aligned}
\lim_{m \to \infty} \frac{m^{\frac{m(b-2)}{2b}}}{b^{\frac{m(b-1)}{b}}} &= \lim_{m \to \infty} e^{\phi(m)} \\
&= \lim_{m \to \infty} e^{\frac{m}{b}} e^{\frac{b-2}{2} \ln m - (b-1) \ln b} \\
&= \lim_{m \to \infty} e^{\frac{m}{b}} \lim_{m \to \infty} e^{\frac{b-2}{2} \ln m - (b-1) \ln b} \\
&= \infty \cdot \infty \\
&= \infty
\end{aligned}
$$

Thus, we have shown that $B_m \in \omega(m^{\frac{m}{2}})$. $\square$

Proof. (Proposition 2.2). The winner determination problem is the same problem as the abstract problem called *weighted set packing* once we view each bid as a set (of items) and the price, $\bar{b}(S)$, as the weight of the set S. The fact that weighted set packing is $\mathcal{NP}$-complete [Karp, 1972] then directly means that winner determination is $\mathcal{NP}$-complete. $\square$

Proof. (Proposition 2.3). Assume for contradiction that there exists a polynomial time approximation algorithm that establishes some bound $k \leq n^{1-\epsilon}$

for the winner determination problem. Then that algorithm could be used to solve the *weighted independent set* problem to the same k in polynomial time.[6] This is because a weighted independent set problem instance can be polynomially converted into a winner determination instance while preserving approximability. This can be done by generating one item for each edge in the graph. A bid is generated for each vertex in the graph. The bid includes exactly those items that correspond to the edges connected to the vertex.

Since the algorithm will k-approximate the weighted independent set problem in polynomial time, it will also k-approximate the independent set problem in polynomial time. A polynomial time k-approximation algorithm for the independent set problem could directly be used to k-approximate the *maximum clique* problem in polynomial time. This is because the maximum clique problem is the independent set problem on the complement graph. But Håstad recently showed that no polynomial time algorithm can establish a $k \leq n^{1-\epsilon}$ for any $\epsilon > 0$ for the maximum clique problem (unless $\mathcal{NP}$ equals probabilistic polynomial time) [Håstad, 1999]. Contradiction. $\square$

Proof. (Proposition 3.1) We first prove that each allocation is generated at most once. The first bullet leads to the fact that an allocation can only be generated in one order of bids on the path. So, for there to exist more than one path for a given allocation, some bid would have to occur multiple times as a child of some node. However, the algorithm uses each bid as a child for a given node only once.

What remains to be proven is that each allocation is generated. Assume for contradiction that some allocation is not. Then, at some point, there has to be a bid in that allocation such that it is the bid with the item with the smallest index among those not on the path, but that bid is not inserted to the path. Contradiction. $\square$

[6] The weighted independent set problem is the problem of finding a maximally weighted collection of vertices in an undirected graph such that no two vertices are adjacent.