Multiagent Coordination Using a Distributed Combinatorial Auction

José M. Vidal
Computer Science and Engineering
University of South Carolina
Columbia, SC 29208
vidal@sc.edu

Abstract

Combinatorial auctions are a great way to represent and solve distributed allocation problems. Unfortunately, most of the winner determination solutions that exists are centralized. They require all agents to send their bids to a centralized auctioneer who then determines the winners. The PAUSE auction, in contrast, is an increasing-price combinatorial auction in which the problem of winner determination is naturally distributed amongst the bidders. Furthermore, the bidders' have an incentive to perform the required computations. But, until now, no bidding algorithm for the auction existed. We provide a bidding algorithm for agents in a PAUSE auction, the PAUSEBID algorithm. It always returns the bid that maximizes the bidder's utility. In effect, PAUSEBID is a the distributed counterpart to the existing centralized winner determination algorithms, from which we borrow several proven techniques. Our test results show that a system where all agents use PAUSEBID finds the revenue-maximizing solution at least 95% of the time. Run time, as expected since this is an NP-complete problem, remains exponential on the number of items.

Introduction

Combinatorial auctions are a popular research topic in part because of their applicability to a large number of distributed allocation problems and multiagent coordination problems (Cramton, Shoham, & Steinberg 2006). However, the bulk of the winner determination algorithms developed thus far are centralized since they assume the standard auction where all the bids are sent to a centralized auctioneer who then runs the winner determination algorithm. Specifically, CASS (Fujishima, Leyton-Brown, & Shoham 1999), CABOB (Sandholm *et al.* 2005), and the earlier Bidtree (Sandholm 2002) all assume this type of centralized auction. Unfortunately, these type of centralized auctions are not a good fit for multiagent systems where computational resources are owned by each agent and each agent has localized information. We need a way of distributing the computation.

Luckily, there do exist auction formulations where the bidders must perform part of the computation, thereby leaving the auctioneer with little or no work to perform. One such auction is the Progressive Adaptive User Selection Environment (PAUSE) auction (Land, Powell, & Steinberg 2006), and earlier and slightly different version of which appeared first in (Kelly & Stenberg 2000), . The PAUSE auction lets bidders distribute the winner determination problem amongst themselves. However, in order to use it in an agent system we first need an algorithm that tells the agents how they are to generate their bids. That is, in the same way that the standard combinatorial auction requires a winner determination algorithm in order to be implemented by an agent system, so does the PAUSE auction require a bidding algorithm for its agents. We thus present the PAUSEBID algorithm which enables agents in a PAUSE auction to find the bids that maximize their utility.

A system of agents using PAUSEBID and the PAUSE auction can effectively and distributively calculate the solution to complex coordination problems. For example, imagine a group of robots trying to pick up and deliver a set of packages in an office building. Each robot is at a different location and has different abilities (some can carry certain types of packages, some can carry multiple packages at a time, etc.) These can decide who will deliver which packages by implementing the PAUSE combinatorial auction where each robot uses its own valuation function for the sets of packages it can deliver. The computation required for calculating the final allocation is naturally distributed among the robots.

The PAUSE Auction

A PAUSE auction for m items has m stages. Stage 1 consists of having simultaneous ascending price open-cry auctions for each individual item. During this stage the bidders can only place individual bids on items. At the end of this state we will know what is the highest bid for each individual good and who placed that bid. In each successive stage $k = 2, 3, \ldots, m$ we hold an ascending price auction where the bidders must submit sets of bids that cover all goods but each one of the bids must be for k goods or less. The bidders are allowed to use bids that other agents have placed in previous rounds when placing their bid, thus allowing them to find better solutions. Also, any new bid set has to have a sum of bid prices which is bigger than the currently winning bid set.

At the end of each stage k all agents know the best bid for every subset of size k or less. Also, at any point in time after stage 1 has ended there is a standing bid set whose value in-

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

creases monotonically as new bid sets are submitted. Since in the final round all agents consider all possible bid sets, we know that the final winning bid set will be one such that no agent can propose a better bid set. Note, however, that this bid set is not guaranteed to be the one that maximizes revenue since we are using an ascending price auction so the winning bid for each set will be only slightly bigger than the second highest bid for the particular set of goods. That is, the final prices will not be the same ones as the prices in a traditional combinatorial auction where all the bidders bid their true valuation. However, there remains the open question of whether the final distribution of goods to bidders found by the PAUSE auction is the same as the distribution dictated by the revenue maximizing solution. Our test results provide an answer to this question.

The PAUSE auction makes the job of the auctioneer very easy. All it has to do is make sure each new bidset adds up to a number that is bigger than the current best as well as make sure that any bids an agent places that are not his do indeed correspond to other agents' bids. The computational problem shifts from one of winner determination to one of bid generation. Each agent must search over the space of all bid sets which contain at least one of its bids. The search is made easier by the fact that the agent need only consider the current best bids and only wants bid sets where its own utility is higher than in the current winning bid. Each agent also has a clear incentive for performing this computation, namely, its utility only increases with each bid set it proposes (of course, it might decrease with the bid sets that others propose). Finally, the PAUSE auction has been shown to be envy-free in that at the conclusion of the auction no bidder would prefer to exchange his allocation with that of any other bidder.

We can even envision completely eliminating the auctioneer and, instead, have every agent perform the task of the auctioneer. That is, all bids are broadcast and when an agent receives a bid from another agent it updates the set of best bids and determines if the new bid is indeed better than the current winning bid. The agents would have an incentive to perform their computation as it will increase their expected utility. Also, any lies about other agents' bids are easily found out by keeping track of the bids sent out by every agent (the set of best bids). Namely, the only one that can increase an agent's bid value is the agent itself. Anyone claiming a higher value for some other agent is lying. The only thing missing is an algorithm that calculates the utility-maximizing bid for each agent.

Related Work A lot of research has been done on various aspects of combinatorial auctions. We recommend (Cramton, Shoham, & Steinberg 2006) for a good review. However, the study of distributed winner determination algorithms for combinatorial auctions is still relatively new. One approach is given by our other algorithms for distributing the winner determination problem in combinatorial auctions (Narumanchi & Vidal 2006), but these algorithms assume the computational entities are the goods being sold and thus end up with a different type of distribution. The VSA algorithm (Fujishima, Leyton-Brown, & Shoham 1999) is another way of performing distributed winner determination in combinatorial auction but it assumes the bids themselves perform the computation. This algorithm also fails to converge to a solution for most cases. In (Parkes & Shneidman 2004) the authors present a distributed mechanism for calculating VCG payments in a mechanism design problem. Their mechanism roughly amounts to having each agent calculate the payments for two other agents and give these to a secure central server which then checks to make sure results from all pairs agree, otherwise a re-calculation is ordered. This general idea, which they call the redundancy principle, could also be applied to our problem but it requires the existence of a secure center agent that everyone trusts. Another interesting approach is given in (Park & Rothkopf 2001) where the bidding agents prioritize their bids, thus reducing the set of bids that the centralized winner determination algorithm must consider, making that problem easier. Finally, in the computation procuring clock auction (Brewer 1999) the agents are given an ever-increasing percentage of the surplus achieved by their proposed solution over the current best. As such, it assumes the agents are impartial computational entities—not the set of possible buyers as assumed by the PAUSE auction.

Problem Formulation

We now introduce some notation to formally describe the problem and our algorithm. Let each bid b be composed of b^{items} which is the set of items the bid is over, b^{value} the value or price of the bid, and b^{agent} the agent that placed the bid. The agents maintain a set B of the current best bids, one for each set of items of size $\leq k$ where k is the current stage. At any point in the auction, after the first round, there will also be a set $W \subseteq B$ of currently winning bids. This is the set of bids that currently maximizes the revenue, where the revenue of W is given by

$$r(W) = \sum_{b \in W} b^{\text{value}}. \tag{1}$$

Agent i's value function is given by $v_i : S \to \Re$ where S is a subset of the items. Given an agent's value function and the current set of winning bids W we can calculate the agent's utility from W as

$$u_i(W) = \sum_{b \in W \mid b^{\text{agent}} = i} v_i(b^{\text{items}}) - b^{\text{value}}. \tag{2}$$

That is, the agent's utility for a bid set W is the value it receives for the items it wins in W minus the price it must pay for those items. If the agent is not winning any items then its utility is zero. The goal of the bidding agents in the PAUSE auction is to maximize their utility, subject to the constraint that their next set of bids must have a total revenue that is at least ϵ bigger than the current revenue, where ϵ is the smallest increment allowed in the auction. Formally, given that W is the current set of winning bids, agent i must find a g^* such that $r(g^*) \geq r(W) + \epsilon$ and

$$g^* = \arg\max_{g \subseteq 2^B} u_i(g), \tag{3}$$

where each g is a set of bids all taken from B and g covers all items.

```
PAUSEBID$(i, k)$
1   $my\text{-}bids \leftarrow \emptyset$
2   $their\text{-}bids \leftarrow \emptyset$
3   for $b \in B$
4       do if $b^{\text{agent}} = i$ or $v_i(b^{\text{items}}) > b^{\text{value}}$
5           then $my\text{-}bids \leftarrow my\text{-}bids$ +new Bid$(i, b^{\text{items}}, v_i(b^{\text{items}}))$
6           else  $their\text{-}bids \leftarrow their\text{-}bids + b$
7   for $S \in$ subsets of $k$ or fewer items such that
            $v_i(S) > 0$ and $\neg\exists_{b \in B} b^{\text{items}} = S$
8       do $my\text{-}bids \leftarrow my\text{-}bids$ +new Bid$(i, S, v_i(S))$
9   $bids \leftarrow my\text{-}bids + their\text{-}bids$
10  $g^* \leftarrow \emptyset$            $\triangleright$ Global variable
11  $u^* \leftarrow u_i(W)$        $\triangleright$ Global variable
12  $h(S) \leftarrow$ max revenue on items from $S$ given $B$, for all $S$.
13  PAUSEBIDSEARCH$(bids, \emptyset)$
14  $surplus \leftarrow \sum_{b \in g^* \mid b^{\text{agent}} = i} b^{\text{value}} - W(b^{\text{items}})$
15  if $surplus = 0$
16     then return $g^*$
17  $my\text{-}payment \leftarrow v_i(g^*) - u^*$
18  for $b \in g^* \mid b^{\text{agent}} = i$
19      do if $my\text{-}payment \leq 0$
20          then $b^{\text{value}} \leftarrow 0$
21          else  $b^{\text{value}} \leftarrow W(b^{\text{items}}) + my\text{-}payment \cdot \frac{b^{\text{value}} - W(b^{\text{items}})}{surplus}$
22  return $g^*$
```

Figure 1: The PAUSEBID algorithm which implements a branch and bound search. i is the agent and k is the current stage of the auction, for $k \geq 2$.

Bidding Algorithm

During the first stage we simply have several English auctions. As such, an agent's dominant strategy is to bid ϵ higher than the current winning bid until it reaches its valuation for that particular item. The only caveat is for agents with sub-additive valuations. These agents must make sure that their valuation for all the subsets they are currently winning is higher than the current sum of the prices. Our algorithm focuses on the succeeding stages: $k > 1$.

Agent i can find g^* by performing a complete search on all the possible combinations of bids within B. This is a large search tree but luckily we can speed up the search by pruning it. We start by noticing that the agent wants to find the set of bids that maximize its revenue and that at any one time there are likely only a few bids within B which the agent can dominate. That is, we start by defining $my\text{-}bids$ to be the list of bids for which the agent's valuation is higher than the current best bid, as given in B. We set the value of these bids to be the agent's true valuation (but we won't necessarily be bidding true valuation, as we explain later). Similarly, we set $their\text{-}bids$ to be the rest of the bids from B. Finally, the agent's search list is simply the concatenation of $my\text{-}bids$ and $their\text{-}bids$. Note that the agent's own bids are placed first on the search list as this will enable us to do more pruning. Lines 3–9 of PAUSEBID, shown in Figure 1, show how we create these lists.

The agent can now perform a branch and bound search on the branch-on-bids tree produced by these bids. This branch and bound search is implemented by PAUSEBIDSEARCH shown in Figure 2. Our algorithm not only implements the standard bound but it also implements other pruning techniques in order to further reduce the size of the search tree.

The bound we use is the maximum utility that the agent can expect to receive from a given set of bids. We call it u^*. Initially, u^* is set to $u_i(W)$ (PAUSEBID line 11) since that is the utility the agent currently receives and any solution he proposes should give him more utility. If PAUSEBIDSEARCH ever comes across a partial solution where the maximum utility the agent can expect to receive is less than u^* then that subtree is pruned (PAUSEBIDSEARCH line 21). Note that we can determine the maximum utility only after the algorithm has searched over all of the agent's own bids (which are first on the list) because after that we know that the solution will not include any more bids where the agent is the winner thus the agent's utility will no longer increase. For example, if an agent has only one bid in $my\text{-}bids$ then the maximum utility he can expect is equal to his value for the items in that bid minus the minimum possible payment we can make for those items and still come up with a set of bids that has revenue greater than $r(W)$. The calculation of the minimum payment is shown in line 19 for the partial solution case and line 9 for the case where we have a complete solution. Note that in order to calculate the $min\text{-}payment$ for the partial solution case we need an upper bound on the payments that we must make for each item. This upper bound is provided by h, defined in PAUSEBID line 12. This upper bound is identical to the one used by the Bidtree algorithm—it merely assigns to each individual item a value equal to the maximum bid in B divided by the number of items in that bid.

The algorithm also uses the h heuristic to prune any branches which cannot lead to a solution with revenue greater than the current W, as shown in lines 16–17 of PAUSEBIDSEARCH. That is, it uses the h function in the same way an A^* algorithm uses its heuristic.

A final pruning technique implemented by the algorithm is ignoring any branches where the agent has no bids in the current answer g and no more of the agent's bids are in the list (PAUSEBIDSEARCH lines 6–7).

The resulting g^* found by PAUSEBIDSEARCH is thus the set of bids that has revenue which is bigger than $r(W)$ and maximizes agent i's revenue. However, agent i's bids in g^* are still set to his own utility and not to the lowest possible price (that is, the min-$payment$). Lines 18–21 in PAUSEBID are responsible for setting the agent's payments so that it can achieve its maximum utility u^*. If the agent has only one bid in g^* then it is simply a matter of reducing the payment of that bid by u^* from the current maximum of the agent's true valuation. However, if the agent has more than one bid then we face the problem of how to distribute the agent's payments among these bids. There are many ways of distributing the payments and there does not appear to be a dominant strategy for performing this distribution. We have chosen to distribute the payments in proportion to the agent's true valuation for each set of goods, as shown in lines 18–21 of PAUSEBID

The PAUSEBID function is called for rounds $k \geq 2$ of the PAUSE auction and it returns the agent's revenue-maximizing bid, if there is one. It assumes that the set of winning bids B and the current best winning bid set W remains constant during its execution.

Analysis

Since PAUSEBID performs a complete branch and bound search for g^* we can prove that it is correct by analyzing its pruning strategies.

Theorem 1. PAUSEBID *finds* g^* *which satisfies* (3) *given a set* B *of current best bids and a currently winning bidset* W.

Proof. The proof follows from the fact that it performs a complete search and only prunes subtrees which are guaranteed to not contain a satisfactory solution. Lines 6–7 of PAUSEBIDSEARCH prune subtrees where the final solution will not contain any bid from the agent thus giving him a utility of zero, lines 16–17 of PAUSEBIDSEARCH prune subtrees where the final solution is guaranteed to have lower revenue than the current solution, and line 21 of PAUSEBIDSEARCH prunes subtrees where the solution is guaranteed to give the agent lower utility than an already found solution. □

We know that in a single-item English auction an agent's myopic best-response strategy is to always bid ϵ higher than the current price as long as his bid is less than his valuation for the item, after which the agent should stop bidding. The PAUSEBID algorithm implements a similar strategy. The agent places the bid which maximizes its own utility and has a revenue greater than the current winning bid. Since the more an agent pays the less utility it receives, the agent always places the bid that has the lowest possible revenue. As such, PAUSEBID implements a myopic best-response bidding strategy given that the agent knows nothing about the others' valuation or bidding strategies.

Unfortunately, PAUSEBID does have certain weaknesses that could be exploited if used against an intelligent opponent who knows the agent is using PAUSEBID. The problem lies in lines 18–21 of PAUSEBID where we distribute the agent's surplus across his bids in g^*. Notice that we distribute the agent's payments *proportionately* to the agent's valuation for that set of items. This has the unfortunate effect of revealing, to some extent, the agent's true relative valuation for the items. For example, if an agent increases his bids for two sets of items, but his increase for the first set is much greater than for the second set then we can deduce that the agent valuates the first set much higher than the second. This knowledge could then, perhaps, be used by a strategic agent to place his own bids. However, based on our previous work on agent modeling (Vidal & Durfee 1998), we believe, that such strategic thinking will incur in large computational costs and will deliver small utility gains. But, even if this belief proves wrong, it is a simple matter to change the surplus distribution method to include some randomness. Of course, even with random distributions, the fact that an agent increases the his bid for certain subsets of items is still a clear signal that its valuation of those subsets is higher than the current price (perhaps, a lot higher?). An opponent might be able to use this knowledge to make better decisions about which sets of items he should bid on.

Because of these strategic issues we cannot claim that the PAUSEBID strategy is a dominant strategy: the best strategy to use regardless of the other agents' strategies. However, we can claim that at each time it is called it returns the bid that maximizes the agent's utility while still having a revenue greater than the current solution and increasing the agent's utility over the one it is currently receiving. Furthermore, as our tests show, if all agents use PAUSEBID then the system as a whole is likely to find the solution that is the same as that found by a centralized winner determination algorithm when everyone reports their true valuations.

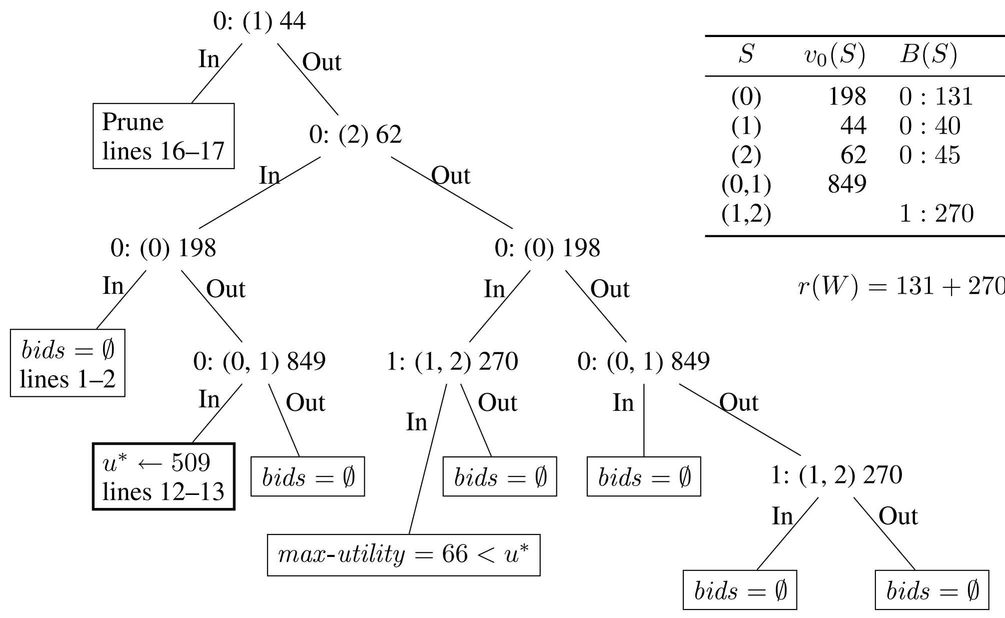
Tests

We have implemented PAUSEBID in order to ensure that it works as predicted and to test how long the auctions take to finish and what is the final solution. In order to do our tests we had to generate value functions for the agents[1]. The algo-

[1]Note that we could not use CATS (Leyton-Brown, Pearson, & Shoham 2000) because it generates sets of bids for an indeterminate number of agents. Its like if you were told the set of bids placed in a combinatorial auction but not who placed each bid or even how many people placed bids, and then asked to determine the value function of every participant in the auction.

```
PAUSEBIDSEARCH($bids, g$)
 1  if $bids = \emptyset$
 2     then return
 3  $b \leftarrow$ first($bids$)
 4  $bids \leftarrow bids - b$
 5  $g \leftarrow g + b$
 6  if $g$ does not contain a bid from $i$
 7     then return
 8  if $g$ includes all items
 9     then $min\text{-}payment \leftarrow \max(0, r(W) + \epsilon - (r(g) - r_i(g)), \sum_{b \in g \mid b^{\text{agent}} = i} B(b^{\text{items}}))$
10          $max\text{-}utility \leftarrow v_i(g) - min\text{-}payment$
11          if $r(g) > r(W)$ and $max\text{-}utility \geq u^*$
12             then $g^* \leftarrow g$
13                  $u^* \leftarrow max\text{-}utility$
14          PAUSEBIDSEARCH($bids, g - b$) $\triangleright$ $b$ is Out
15     else $max\text{-}revenue \leftarrow r(g) + h$(items not in $g$)
16          if $max\text{-}revenue \leq r(W)$
17             then PAUSEBIDSEARCH($bids, g - b$) $\triangleright$ $b$ is Out
18          elseif $b^{\text{agent}} \neq i$
19             then $min\text{-}payment \leftarrow r(W) + \epsilon - (r(g) - r_i(g)) - h$(items not in $g$)
20                  $max\text{-}utility \leftarrow v_i(g) - min\text{-}payment$
21                  if $max\text{-}utility > u^*$
22                     then PAUSEBIDSEARCH($\{x \in bids \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}, g$) $\triangleright$ $b$ is In
23                  PAUSEBIDSEARCH($bids, g - b$) $\triangleright$ $b$ is Out
24          else
25                  PAUSEBIDSEARCH($\{x \in bids \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}, g$) $\triangleright$ $b$ is In
26                  PAUSEBIDSEARCH($bids, g - b$) $\triangleright$ $b$ is Out
27  return
```

Figure 2: The PAUSEBIDSEARCH recursive procedure where $bids$ is the set of available bids and g is the current partial solution.



S	$v_0(S)$	$B(S)$	in W?	$h(S)$
(0)	198	0 : 131	Yes	131
(1)	44	0 : 40	No	135
(2)	62	0 : 45	No	135
(0,1)	849		No	NA
(1,2)		1 : 270	Yes	NA

$$r(W) = 131 + 270 = 401$$

Figure 3: Sample search tree produced by PAUSEBIDSEARCH for agent 0 given the values on the table at the top right. We assume that $\epsilon = 1$. The nodes are bids of the form "agentid : (items) price".

```
GENERATEVALUES($i$, $items$)
1  for $x \in items$
2      do $v_i(x) = \text{EXPD}(.01)$
3  for $n \leftarrow 1 \ldots (num\text{-}bids - items)$
4      do $s_1, s_2 \leftarrow$Two random sets of items with values.
5          $v_i(s_1 \cup s_2) = v_i(s_1) + v_i(s_2) + \text{EXPD}(.01)$
```

Figure 4: Algorithm for the generation of random value functions. EXPD(x) returns a random number taken from an exponential distribution with mean $1/x$.

rithm we used is shown in Figure 4. The type of valuations it generates correspond to domains where a set of agents must perform a set of tasks but there are cost savings for particular agents if they can bundle together certain subsets of tasks. For example, imagine a set of robots which must pick up and deliver items to different locations. Since each robot is at a different location and has different abilities, each one will have different preferences over how to bundle. Their costs for the item bundles are subadditive, which means that their preferences are superadditive.

The first tests we performed simply ensured the proper functioning of the algorithm. We then compared the solution found by our algorithm to the solution found by CASS when given a set of bids that corresponds to the agents' true valuation. That is, for each agent i and each set of items S for which $v_i(S) > 0$ we generated a bid. This set of bids was fed to CASS which implements a centralized winner determination algorithm to find the solution which maximizes revenue. When we compared this solution with the set of bids found by PAUSEBID we found that on at least **95% of the runs** both algorithms arrive at the same solution. Specifically, with 5 bidders, 6 items, and 1000 runs, we found that on 96.2% of the runs both algorithms arrived at the same solution. Note, however, that the revenue from the PAUSE auction on all the auctions is always smaller than that found by CASS using the agents' valuations. Since PAUSE uses English auctions the final prices (roughly) represent the second-highest valuation, plus ϵ, for that set of items.

The cases where we failed to arrive at the revenue of the revenue-maximizing solution are those where there was a large gap between the first and second valuation for a set (or sets) of items. If the revenue-maximizing solution contains the bid (or bids) using these higher valuation then it is impossible for the PAUSE auction to find this solution because that bid (those bids) is never placed. For example, if agent i has $v_i(1) = 1000$ and the second highest valuation for (1) is only 10 then i only needs to place a bid of 11 in order to win that item. If the revenue-maximizing solution requires that 1 be sold for 1000 then that solution will never be found because that bid will never be placed.

We are also interested in the real-time performance of the system. We define a time unit as the time it takes for all agents to place a bid. We can then measure how many time units it takes for the system to arrive at the final solution.

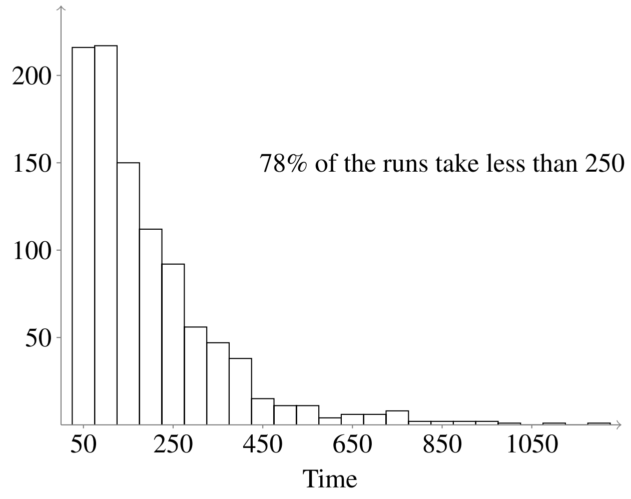


Figure 5: Distribution of the times it took to run each auction, for 1000 runs with 6 agents and 5 items. The y-axis is the number of runs that took at most x time units. A time unit consist of all agents having a chance to place a bid.

Figure 5 shows a distribution of the time it took for each one of 1000 runs for the system to finish. As we expected, the distribution is thick on the left side (short time) but has a long tail towards the right. This shape is similar to the exponential distribution from which the agent's valuations were taken. The long times are from those cases where two or more agents happen to have very high valuations for the same set of items and engage in the typical oneupmanship seen in English auctions.

The scalability of the algorithm can be determined by counting the number of times that PAUSEBIDSEARCH gets invoked for each time that PAUSEBID is called, that is, the number of nodes expanded in the search tree. Figure 6 shows the average number of nodes expanded on each invocation of PAUSEBID as we vary the number of items for sale. As expected since this is an NP-complete problem, the number of nodes does grow exponentially with the number of items. But, the actual number of nodes is a much smaller than the worst-case scenario of x^x where x is the number of items. For example, for 10 items we expand slightly less than 10^4 nodes which is much smaller number than 10^{10}. Notice also that our value generation algorithm (Figure 4) generates a number of bids that is exponential on the number of items, as might be expected in many situations. As such, these results do not support the conclusion that time grows exponentially with the number of goods when the number of bids is independent of the number of goods. We expect that PAUSEBID will grow exponentially as a function the number of *bids*, but stay roughly constant as the number of items grows.

Future Work

This algorithm continues our research in distributed winner determination algorithms for combinatorial auctions (Narumanchi & Vidal 2006). In contrast with our previous work, with the PAUSE auction we have made the assumption that

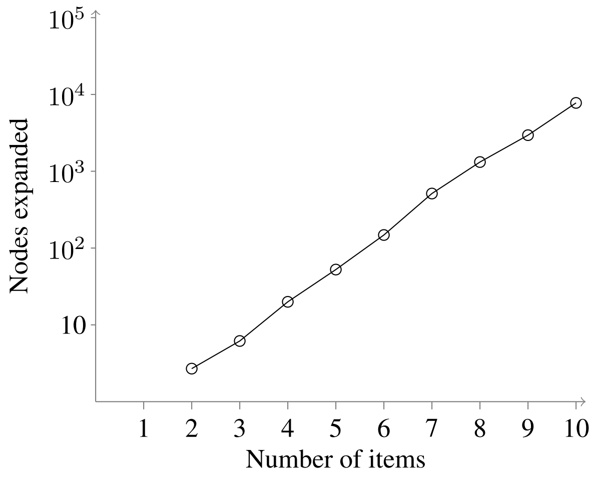


Figure 6: Average number of nodes expanded as a function of the number of items in the auction. There were 5 agents in this experiment.

the agents are the buyers and each one has multiple bids that it wants to place.

There are many obvious ways to improve on the performance of PAUSEBID. The most dramatic gain will probably be when we modify it to cache partial solutions. As it is, the algorithm performs each search completely from scratch each time it is invoked. However, since these are English auctions where each agent submits, at most, one bid set then it is likely that B does not change much from time t to $t+1$. We will be implementing caching techniques similar to those used by CABOB, where the algorithm remembers the best bid set for each set of items previously searched over. The added complication we face is that we must come up with an efficient scheme for invalidating the proper entries in the cache when B is updated.

Other possible improvements include developing ways that agents may cooperate in order to minimize any redundant work (while still not giving them any incentive to cheat), ways of speeding up the inherent real-time slowness of the English auction, exploiting the fact that in the k level of the auction any new bid set is likely to include at least one bid of size k, and eliminating the need for agents to constantly broadcast new bids and instead use a multicasting method.

Conclusion

We have presented PAUSEBID—an algorithm for bidding in a PAUSE auction that is guaranteed to find the bid which maximizes the agent's utility given the outstanding best bids. Agents in a multiagent system can use PAUSEBID to implement a distributed combinatorial auction and thereby solve complex coordination problem distributively. The agents can even be selfish as the system provides an incentive for them to perform the computations. As it is an NP-complete problem, the running time of our algorithm remains exponential but it is significantly better than a full search. We are currently working on caching techniques that should dramatically improve the performance of the algorithm. Centralized combinatorial auctions are only of limited use for building multiagent systems, we believe that distributed algorithms for achieving similar coordination will be much more relevant to this domain.

References

[Brewer 1999] Brewer, P. J. 1999. Decentralized computation procurement and computational robustness in a smart market. *Economic Theory* 13(1):41–92.

[Cramton, Shoham, & Steinberg 2006] Cramton, P.; Shoham, Y.; and Steinberg, R., eds. 2006. *Combinatorial Auctions*. MIT Press.

[Fujishima, Leyton-Brown, & Shoham 1999] Fujishima, Y.; Leyton-Brown, K.; and Shoham, Y. 1999. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 548–553. Morgan Kaufmann Publishers Inc.

[Kelly & Stenberg 2000] Kelly, F., and Stenberg, R. 2000. A combinatorial auction with multiple winners for universal service. *Management Science* 46(4):586–596.

[Land, Powell, & Steinberg 2006] Land, A.; Powell, S.; and Steinberg, R. 2006. PAUSE: A computationally tractable combinatorial auction. In Cramton et al. (2006). chapter 6, 139–157.

[2000] Leyton-Brown, K.; Pearson, M.; and Shoham, Y. 2000. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, 66–76. ACM Press. `http://cats.stanford.edu`.

[2006] Narumanchi, M. V., and Vidal, J. M. 2006. Algorithms for distributed winner determination in combinatorial auctions. In *LNAI volume of AMEC/TADA*. Springer.

[2001] Park, S., and Rothkopf, M. H. 2001. Auctions with endogenously determined allowable combinations. Technical report, Rutgets Center for Operations Research. RRR 3-2001.

[2004] Parkes, D. C., and Shneidman, J. 2004. Distributed implementations of vickrey-clarke-groves auctions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems*, 261–268. ACM.

[2005] Sandholm, T.; Suri, S.; Gilpin, A.; and Levine, D. 2005. CABOB: a fast optimal algorithm for winner determination in combinatorial auctions. *Management Science* 51(3):374–391.

[2002] Sandholm, T. 2002. An algorithm for winner determination in combinatorial auctions. *Artificial Intelligence* 135(1-2):1–54.

[1998] Vidal, J. M., and Durfee, E. H. 1998. Learning nested models in an information economy. *Journal of Experimental and Theoretical Artificial Intelligence* 10(3):291–308.