LNAI volume of AMEC/TADA, 2006.

Algorithms for Distributed Winner Determination In Combinatorial Auctions

Muralidhar V. Narumanchi and José M. Vidal

Computer Science and Engineering
University of South Carolina
Columbia, SC. 29208
narumanc@cse.sc.edu, vidal@sc.edu

Abstract. The problem of optimal winner determination in combinatorial auctions consists of finding the set of bids that maximize the revenue for the sellers. Various solutions exist for solving this problem but they are all centralized. That is, they assume that all bids are sent to a centralized auctioneer who then determines the winning set of bids. In this paper we introduce the problem of distributed winner determination in combinatorial auctions which eliminates the centralized auctioneer. We present a set of distributed search-based algorithms for solving this problem and study their relative tradeoffs.

1 Introduction

In a combinatorial auction the buyers bid on bundles of items. After clearing, each buyer receives either the entire bundle he bid on or nothing. Combinatorial auctions are often preferred over sequential auctions because bidders can express complementarity and substitutability of their choices within the bids. The optimal winner determination problem in a combinatorial auction involves finding the set of bids that maximizes the revenue generated. This problem is known to be an $\mathcal{NP}$-Hard problem [1]. Various centralized approaches using A^* [2], dynamic programming [1], integer programming [3], linear programming [4], and approximation techniques [5] have been proposed for determining the optimal and approximately-optimal solution. All these algorithms assume the existence of a centralized auctioneer who collects all the bids and computes the set of winning bids. All these algorithms also fail to address the question of revenue division amongst the winning goods. In this paper we investigate the problem of distributed winner determination, that is, the determination of the set of winning bids in the absence of a centralized auctioneer. We provide some distributed search-based solutions as well as a negotiation-based approach which also performs revenue division.

Our research is motivated by a vision of a future Internet-based distributed electronic marketplace. The system would be a peer-to-peer system, without the need for a centralized auctioneer, and would have to provide the proper incentives for selfish agents to participate and perform their duties as required. For such a system to exist we will need, among other technologies, protocols that distribute the computational task of winner determination. But, since the agents performing the computation have an interest in the outcome of the computation and might try to manipulate it, we need protocols that provide the correct incentives to agents and prevent them from manipulating the outcome. As such, our problem is an instance of a distributed algorithmic mechanism design problem [6,7]. Hence it is also essential that we address the issue of revenue distribution among the sellers in each winning combinatorial bid, an issue that has not been addressed by any of the centralized winner determination algorithms we have found.

Specifically, we assume that each agent has a good for sale and receives combinatorial bids from prospective buyers. The agents must then implement some protocol which will lead to the distributed calculation of the set of winning bids. We consider both the case where the agents are cooperative and when they are selfish. A system with cooperative agents could arise if all the agents are owned by the same entity or if the participants have previously arrived at an off-line agreement. Selfish agents more closely simulate the selfish interests of their human counterparts who want to maximize their profit.

We start by covering some of the past work on combinatorial auctions in section 2. Section 3 formally defines the distributed combinatorial auctions problem. Sections 4, 5 and 6 give a complete algorithm, a simple hill-climbing algorithm and a partitioning-based algorithm, respectively. Section 7 provides a negotiation based approach. Finally, section 8 shows our test results and section 9 discusses the future work.

2 Related work

Sandholm [2] has given an algorithm for calculating the optimal set of bids in a combinatorial auction using an implementation of A^*. Hoos and Boutilier have provided a solution using stochastic local search [8]. Rothkopf et al. provides a solution using dynamic programming [1]. Fujishima et al. proposes one method to speed up the search by structuring the search space and a heuristic method that lacks optimality guarantees but performs well on average [9]. All these algorithms are centralized.

In the area of multiple agents operating simultaneously in a market setting, Preist provides an algorithm for agents that participate in multiple English auctions [10,11]. Wellman *et. al.* [12] use a market mechanism to solve a decentralized scheduling problem. Both solve different problems from ours. The reader new to combinatorial auctions can read the survey provided by [13].

3 Problem Description

A distributed combinatorial auction is defined as a set goods G where $g_i \in G$ and $|G| = n$, a set of consumers C where $c_i \in C$ and $|C| = k$, and a set of bids B where $b_i \in B$. Each bid b_i is a tuple $\{c, g, p\}$ where c is the consumer who placed the bid, $g \subseteq G$ is the set of goods being bid on, and p is the bid price. There is no centralized auctioneer who collects these bids. We will use b_i^g to refer to the set of goods for bid b_i, b_i^p to refer to the price of bid b_i and g_i^b to refer to the list of bids in which good g_i is present.

Each consumer can place any number of bids[1]n our paper, we assume that each consumer places a single bid.. The bid can be for either a single good or a combination of goods. For example, consumer c_i can place a bid b_k on the bundle $\{g_1, g_4, g_7\}$ for a value of v_1 and another bid b_j on bundle $\{g2\}$ for value v_2.

Definition 1 (Feasible Allocation). *An allocation A of goods is a feasible allocation if and only if no two bids in the allocation share a good.*

The set of all feasible allocations, given B, is given by F which is

$$F \equiv \{b \subseteq B \,|\, \forall_{b_i, b_k \in A, i \neq k} b_i^g \cap b_k^g = \emptyset\}. \tag{1}$$

The value of an allocation A is given by

$$V(A) = \sum_{b \in A} b^p. \tag{2}$$

The revenue maximizing solution A^* is the feasible allocation that maximizes the total price paid for all the goods, that is $A^* = \arg\max_{A \subseteq F} V(A)$.

In distributed combinatorial auctions there is no centralized auctioneer who collects all the bids. Instead, we assume that each good for sale is represented by an agent. When a consumer places a bid b_i, the bid is passed on to b_i^g which are the agents representing the goods present in the bids[2]. Any agent g_i can communicate with any other agent. Thus each agent has the list of bids in which it is present.

We further assume that a bid can be cleared if and only if all the agents in the bid agree to clear it. The final agreement reached by the agents is final and binding. We also assume that the agents don't have a reservation price for their goods and that goods can be sold only once. Finally, some of the algorithms we will introduce make use of $w(b)$ which is the average price per good for bid b. That is, $w(b) = \frac{b^p}{|b^g|}$.

The question we try to answer is: *How can the agents determine the set of winning bids in the absence of the centralized controller?*

4 Complete Search Algorithm

In a complete search the agents search over the space of all the possible allocations to determine the optimal allocation. Since there is no centralized auctioneer

[1] i

[2] In this paper we use the terms "agent" and "good" interchangeably.

```
COMPLETE-SEARCH(allocation-from-parent)
 1  $global\text{-}utility \leftarrow 0$
 2  $final\text{-}allocation \leftarrow \emptyset$
 3  $cleared\text{-}goods \leftarrow \bigcup_{b \in allocation\text{-}from\text{-}parent} b^g$
 4  $valid\text{-}bid\text{-}pool \leftarrow \{b \in bid\text{-}pool \mid \forall_{g \in b^g} b \notin cleared\text{-}goods\}$
 5  if $child = \emptyset$        ▷ I am leaf.
 6     then $final\text{-}allocation \leftarrow allocation\text{-}from\text{-}parent \cup \arg\max_{b \in valid\text{-}bid\text{-}pool} w(b)$
 7          return $final\text{-}allocation$
 8  if $valid\text{-}bid\text{-}pool = \emptyset$
 9     then $final\text{-}allocation \leftarrow child$ .COMPLETE-SEARCH($allocation\text{-}from\text{-}parent$)
10     else
11          for $bid \in valid\text{-}bid\text{-}pool$
12              do $new\text{-}allocation \leftarrow allocation\text{-}from\text{-}parent \cup bid$
13                 $allocation\text{-}from\text{-}successor \leftarrow child$ .SUCCESSOR($new\text{-}allocation$)
14                 if $V(allocation\text{-}from\text{-}successor) > V(global\text{-}utility)$
15                    then $global\text{-}utility \leftarrow V(allocation\text{-}from\text{-}successor)$
16                         $final\text{-}allocation \leftarrow new\text{-}allocation$
17  return $final\text{-}allocation$
```

Fig. 1. Complete search algorithm. It is started by calling the root agent with COMPLETE-SEARCH($\emptyset$).

that has global information, the agents must pass messages to each other and perform the search in a distributed manner.

In this algorithm we assume that the agents possess a linear ordering such that every agent has a *child* variable which points to its child, except for the leaf node who sets this variable to $\emptyset$. Each agent also maintains the following variables:

- *bid-pool* is the list of bids in which it is present,
- *final-allocation* is the best allocation encountered thus far in the execution,
- *global-utility* is the utility of the final-allocation,

Each agent adds zero-valued singleton bid for itself, even if a singleton bid is present in the list of bids. This bid enables the agent to search for the allocations where the agent is not cleared in any bid. The head agent (whose execution is initiated by the controller) does not have any parent. Similarly the last agent in the ordering does not have a child agent so it does not send a message to child or wait for a reply. The agents search all the possible allocations to determine the optimal winner, see Figure 1.

We can prove the correctness of this algorithm by observing that the algorithm is performing a linear search of all the feasible allocations[3]. The agents simply implement a depth first search over all possible bid sets except that they

[3] Proofs omitted due to lack of space

```
Cleared(sender)
1  $\textit{list-of-bids} \leftarrow \{b \in \textit{list-of-bids} \mid \textit{sender} \notin b^g\}$
2  if $\textit{list-of-bids} = \emptyset$
3      then Exit      ▷ We are done. I did not sell my good.
4  Send-Accept()

Accept(sender, bid)
1  $\textit{accepted}[\textit{bid}] \leftarrow \textit{accepted}[\textit{bid}] \cup \textit{sender}$
2  if $\textit{accepted}[\textit{best-bid}] = \textit{best-bid}^g$     ▷ Everyone has accepted it.
3      then for $\textit{agent} \in \textit{neighbors}$
4                do $\textit{agent}$ .Cleared($g_i$)
5            Exit      ▷ We are done. I sold my good.

Send-Accept()
1  if $\textit{best-bid} \notin \textit{list-of-bids}$
2      then $\textit{best-bid} \leftarrow \arg\max_{b \in \textit{list-of-bids}} w(b)$
3            $\textit{accepted}[\textit{best-bid}] = \textit{accepted}[\textit{best-bid}] \cup g_i$
4            for $\textit{agent} \in \textit{best-bid}^g$
5                do $\textit{agent}$ .Accept($g_i$, $\textit{best-bid}$)

Hill-Climbing()
1  $\textit{list-of-bids} \leftarrow g_i^b$
2  $\textit{neighbors} \leftarrow \bigcup_{b \in g_i^b} b^g$
3  $\textit{best-bid} \leftarrow \emptyset$
4  Send-Accept()
```

Fig. 2. Hill-Climbing algorithm. It is started by having all agents execute Hill-Climbing.

only check feasible bid sets. As such, this algorithm sequentializes the agents' execution so it has a long running time.

5 Individual Hill-Climbing Algorithm

We now present an algorithm that creates a feasible allocation using a simple hill-climbing approach. In this approach, the agents simply clear bids in a greedy fashion ordered by $w(b)$, the average value per good until there are no more bids that can be cleared. In fact, this algorithm is but a variation of the algorithm given in [14] for coalition formation.

The algorithm proceeds as follows: Each agent finds the bid in its *list-of-bids* which has the highest average value. The agent then sends an Accept message to the goods that are present in this bid. The agent clears this bid only when it receives an Accept message from all the goods in this bid. This ensures that a bid is cleared if and only if all the goods in the bid agree to clear it. When an agent clears a bid, it sends a CLEARED to all its neighbors—the set of agents with which it shares some bid—telling them that it has cleared and all bids including the agent should be dropped from consideration. The agents that receive a CLEARED message from agent *sender* delete the bids $sender^b$. The agents stop execution when they clear a bid or the *list-of-bids* is empty. See Figure 2 for the complete algorithm.

It is easy to show that this algorithm always finds a feasible allocation and never enters a deadlock as it only considers feasible solutions. However, the algorithm is not guaranteed to converge to the global optimal allocation as it can get stuck at a local maxima by clearing a bid that has high $w(b)$ but is not to be found in the optimal allocation.

6 Partitioning based search

The greedy algorithm produces a non-optimal solution in polynomial time and the complete search provides the optimal solution in exponential time. Although the complete algorithm determines the optimal winner in a distributed manner, there is no parallelism as only one agent is active at any instant. The agents in the hill-climbing algorithm execute in parallel but they can get stuck at local maxima. Hence we now present a partitioning based approach. Our main motivation for proposing this approach is to obtain solutions whose quality is better than solutions produced by greedy approach but where the execution is comparable to the time taken by the greedy algorithm. In this approach, we partition the goods and the agents perform a complete search within the group (while ignoring the bids outside the partition). The algorithm proceeds as follows:

1. The controller partitions the agents into different groups. The controller also selects the *headAgent* of every group.
2. Each agent is provided with its partition information (the linear ordering in its partition).
3. Each agent deletes the bids that contain any good not present in its partition.
4. The controller initiates the COMPLETE-SEARCH algorithm in every partition.

In order to explain how the controller partitions the agents, we first define the following:

Definition 2 (Graphical representation of Combinatorial Auction). *A combinatorial auction can be represented as a graph $G = (N, E)$, where N is the set of nodes and E is the set of edges. Each node corresponds to a good on sale. An edge exists between any two nodes if they are present in the same bid.*

It is not always possible to divide the goods into disjoint partitions (where there is no edge between partitions). There could be bids on goods in different partitions. Currently, we use a greedy approach to address this issue. The agents do not consider the bids that have a good that is not present in its partition. This approach will result in an optimal solution only if the ignored bids are not part of the optimal allocation.

```
CLEARED($j$)
1  $list\text{-}of\text{-}bids \leftarrow \{b \in list\text{-}of\text{-}bids \mid j \notin b\}$
2  UPDATE-BEST-BID

READY($j, bid$)
1  $ready[j] \leftarrow bid$
2  if $\forall_{g \in bid^g}\ ready[g] = bid$
3      then for $g \in neighbors$
4               do $g$.CLEARED($i$)
5           Exit      ▷ Cleared my good with $bid$.

TELL-ASK-VALUE($j, val$)
1  $ask\text{-}value[j] \leftarrow val$
2  UPDATE-BEST-BID()
```

Fig. 3. Modified MCP message receiving procedures.

7 Negotiation Based Approaches

The search-based approaches provided in the earlier sections ignored the issue of splitting revenue among multiple sellers. That is unrealistic in cases where, for example, one agent has a good that is in much more demand than all the other goods in the combinatorial bids that it is in. This problem has been identified for a long time by sociologists studying social networks [15], and by economics studying social networks [16] (note that their networks are different, even if they refer to them by the same name). We use two approaches for addressing the issue of revenue division. The first technique is inspired from the well-studied monotonic-concession protocol [17] and in the second approach we borrow results from sociological network exchange theory [15].

7.1 Modified Monotonic Concession Protocol

In this section we propose a modified version of the Monotonic Concession Protocol (MCP). In MCP the two negotiating nodes alternately propose a deal that allocates the revenue between the agents. A deal d consists of the tuple (p_1, p_2) such that $p_1 + p_2 = b^p$, where p_1 is the amount agent 1 gets and p_2 is what agent 2 gets. If the receiving node gets an offer where it gets more than or equal to what it had asked for in the last round the protocol terminates. If the receiving node does not agree to the offer, in next round it should propose a new deal, subject to the condition that its payment for the other agent must be strictly higher than in the previous deal. This protocol will either converge to a solution or terminate without agreements if time runs out. Unfortunately, MCP can only be used for bi-party negotiation.

We propose a modified-MCP (mMCP) that can be used for simultaneous multi-party negotiation for the division of the revenue. In it, each agent maintains

```
UPDATE-BEST-BID()
1   for $b \in$ list-of-bids
2       do $demand[b] \leftarrow \sum_{g \in b^g}$ ask-value$[g]$
3   ready-to-clear $\leftarrow \{b \in$ list-of-bids $\mid demand[b] \leq b^p\}$
4   if ready-to-clear $\neq \emptyset$
5     then
6         Sort ready-to-clear first by demand and second by bid id.
7         best-bid $\leftarrow$ first(ready-to-clear)
8         for agent $\in$ best-bid$^g$
9             do agent.READY($i$, best-bid)
10

mMCP()
1   for $j \in G$
2       do ask-value$[j] \leftarrow \infty$
3   neighbors $\leftarrow \bigcup_{b \in g_i^b} b^g$
4   list-of-bids $\leftarrow g_i^b$
5   for ask-value$[i] \leftarrow \max_{b \in g_i^b} b^p$ to 0 step 1
6       do for agent $\in$ neighbors
7              do agent.TELL-ASK-VALUE($i$, ask-value$[i]$)
8          Wait for all neighbors to tell me their ask-value
9   Exit    ▷ Unable to clear my good.
```

Fig. 4. Modified MCP main procedures.The algorithm starts by having all agents execute mMCP.

an *ask-value* which is initialized to the maximum the agent can expect to get given the bids it is involved in. The algorithm then proceeds as follows:

1. At each time-step, the agents send their *ask-value* to other agents with which it is involved in negotiation. As in MCP, the agents have to reduce their *ask-value* from what they demanded in the previous round.
2. Upon receiving the *ask-value* of its neighbors the agent checks if it can still get its *ask-value* for the bids in which it is present, that is, $\sum_{i \in b^g} ask\text{-}value_i \leq b^p$.
3. If the agent can get its *ask-value* on any bid, it clears the bid and it informs the other agents with which it is negotiating that it is out of negotiations.

Just like in MCP, since every node has to lower its *ask-value* in successive iterations, the nodes converge to a solution. However, a problem with the mMCP is that it can cause some revenue to be left unallocated, which happens when the revenue is not evenly divisible by the number of participants given the decrement step (which is 1 in the algorithm as shown but can be set to any positive constant). See Figures 3 and 4 for the detailed algorithm.

7.2 Sociological network exchange theory

Sociological Network Exchange Theory (NET) studies the effects of power on the outcomes of exchanges between people in power relation-networks. In a network, the nodes are the participants and any two nodes can negotiate (for dividing a resource or exchanging goods) if they have an edge between them. The edges represent the amount the agents are trying to divide. Based on extensive studies with human subjects, Sociologists have been able to identify equations that can predict the outcome of human negotiations in particular networks.

Specifically, in [15, Chapter 2] Willer presents an equation which predicts that an exchange occurs on any relation between two nodes at equi-resistance. For example, if nodes A and B want to divide some resource between them then the amounts that A and B will agree on (P_A and P_B respectively) can be obtained by solving equations (3) and (4) for P_A and P_B. In these equations, P_A^{con} is the amount that A makes if it has a confrontation with B (i.e., it doesn't exchange with B) and P_A^{max} is the maximum that A can make from exchange with B.

$$\frac{P_A^{max} - P_A}{P_A - P_A^{con}} = \frac{P_B^{max} - P_B}{P_B - P_B^{con}} \tag{3}$$

$$P_A + P_B = \mathit{TotalRevenue} \tag{4}$$

Equation (3) tells us that the resistance of A must be equal to the resistance of B. Equation (4) tells us that the sum of the payments must be equal to the total revenue. We can easily generalize these equations to n agents by simply adding another resistance equation for each agent and insisting that all resistances must be the same[4]. In all cases we end up with $n+1$ equations of n variables, so we can solve for the payments.

The iterated equi-resistance method [15] tells us to start out with initial payments for the agents equal to an even distribution of the total revenue and then iteratively solve the resistance equations for each agent in order to find its payment given those of the other agents. We are to continue doing this for several rounds or until the system stabilizes. At some point, the agents decide to take the deal (bid) for which they are to receive the highest payment.

This method is easy to implement in a simulator. All we need to do is at each time step calculate the agents' payments by solving the equi-resistance equations. We can then continue to do this until either the payments stabilize or we detect that they have entered a cycle. This type of implementation is the one we have used for the test results in section 8.

We do not yet have a distributed algorithm that can implement this method. One problem is the fact that the calculation of an agent's payments requires

[4] However, we must stress that studies with human subjects only consider binary negotiations. As such, there is no empirical evidence to suggest that human behavior can be predicted using the equi-resistance equation for negotiations among three or more agents.

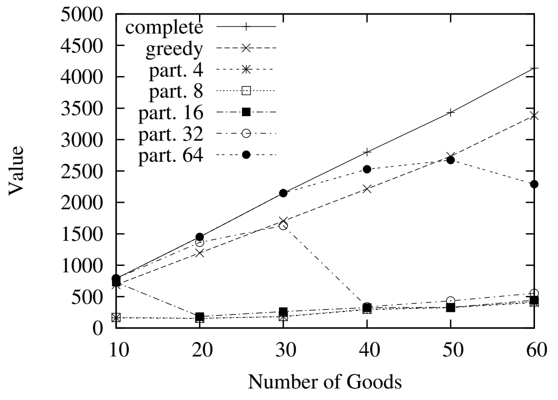

Fig. 5. Value of the winning set of bids.

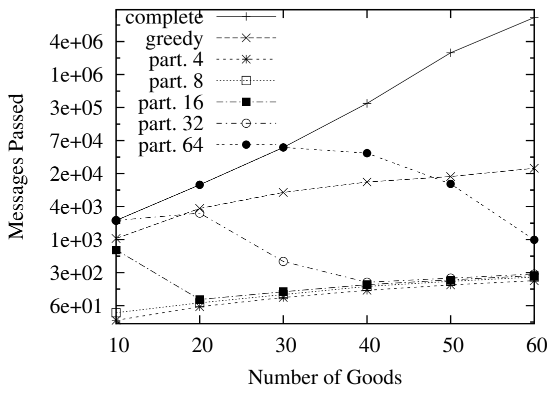

Fig. 6. Total number of messages sent.

knowledge of the P^{con} values for all the agent's neighbors. It is unclear to us how an agent might come to acquire this knowledge if we assume that all agents are selfish. Still, since this method is predicts the behavior of humans, who do not know their opponents' P^{con} values, we are confident that we will come up with an appropriate distributed algorithm in the near future.

8 Preliminary Results

The input data for our algorithms was generated using CATS [18] using random distribution. Figure 5, compares the value of allocation for the three search algorithms. The first point on x-axis consists of 10 goods and 50 bids (thereafter the goods and bids increment by 5 and 50 respectively). The values shown in the figure are the average of 25 runs. As expected, the complete algorithm computes

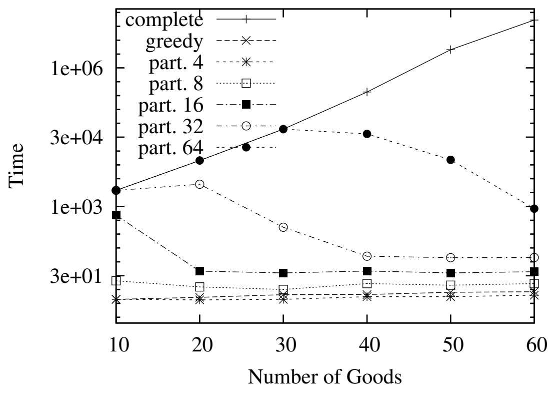

Fig. 7. Total time spent by algorithm.

the solution with best revenue. Similarly, figures 6 and 7 compare the messages passed and the execution time (in clock ticks[5]) respectively for the three search algorithms. The complete search algorithm takes exponential time $O(n^{|b|})$ to provide the optimal allocation. The greedy algorithm takes linear time $O(|b|)$, where n is the number of goods and $|b|$ is the number of bids.

Figure 8 shows the value of the allocations produced by mMCP and iterated-resistance equations. The simulations were run on randomly generated data on NetLogo [19]. In each run, the protocol and the iterations were run for 10 cycles. Even though our solution using resistance equations is not guaranteed to converge, the results are very promising because the algorithm seems to produce high-valued allocations (even though suboptimal) for many cases and always in a short period of time.

9 Discussion and Future Work

We have presented the new problem of distributed winner determination in combinatorial auctions which is an instance of a distributed search problem and, when selfish agents are assumed, is an instance of a distributed algorithmic mechanism design problem. We presented and compared several algorithms for solving the problem under various circumstances. Our results are summarized in Table 1.

The complete algorithm performs a linear search to determine the optimal winner. This algorithm works in the absence of a centralized auctioneer. However, the running time will be of the order of total number of feasible allocations. This is because, even though it is distributed, the agents do not execute in parallel. At any given time only one agent is performing the computation. The hill climbing

[5] The clock tick was chosen to be long enough for the agents to process and execute a single iteration of the search algorithm.

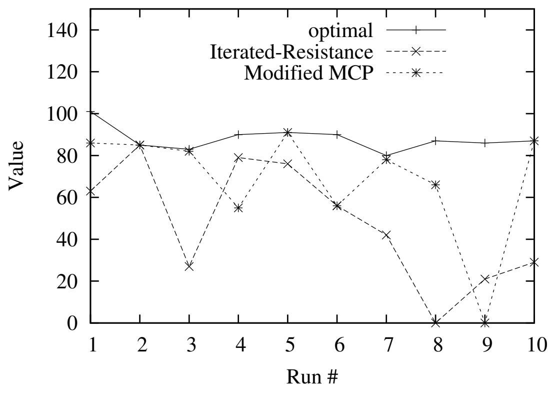

Fig. 8. Value of the solution found by mMCP, iterated-resistance equations, and the global optimum.

Table 1. Algorithm Comparison

Algorithm	Optimal?	Agents	Time	Revenue Split?	Always Converges?
Complete Search	Yes	Cooperative	Exponential	No	Yes
Hill Climbing	No	Cooperative	Linear: $O(\|b\|)$	No	Yes
Partitioning	No	Cooperative	Dependent on partition size	No	Yes
mMCP	No	Selfish	Linear: $O(\|b\|)$	Yes	Yes
Equiresistance	No	Selfish	Not defined	Yes	No

algorithm on the other hand does not guarantee an optimal solution but performs much faster. More tests need to be done in order to determine the expected quality of the solution found by the hill climbing algorithm.

The partitioning based approach ignores the bids outside partitions. This results in sub-optimal solution if the ignored bids are part of the optimal solution. One of the reasons our partitioning approach performed worse than hill-climbing (fig. 5) is that the goods were randomly partitioned. One way to improve the quality of solution in the general case would be to partition the strongly connected goods together, i.e., try to put goods that have lot of common bids in one partition. We intend to extend our work to create dynamic distributed partitioning algorithms that can tailor their partitioning strategy to the characteristics of the set of bids under consideration.

The mMCP that we proposed is again similar to a greedy approach. It can converge to a local maximum and is always guaranteed to converge. We are currently testing it to see if we can predict what are the characteristics of the solution that it converges to. We are also studying modifications of the algorithm that force convergence to optimal as well as the tradeoffs associated with using different step sizes and other shortcuts for faster convergence.

Our study of the applicability of the NET equations is preliminary. We note that these equations can be applied only if agents' know their neighbors' P_{con} values—an unrealistic assumption in most cases. Another problem we face is the fact that the algorithm does not always converge. We are studying possible ways of either forcing convergence or determining a priori if the problem is one that will converge. Still, we are attracted to the fact that the equi-resistance equations have been shown to model the behavior of humans. We believe that the widespread adoption of a peer-to-peer agent-based marketplace requires agents that behave as humans. That is, if a user notices his agent either gives up negotiation too soon or is too aggressive in its negotiations then the user will likely not use that system. We see the possibility of a whole research program dedicated to building agents that negotiate, not necessarily optimally, but as humans would.

In summary, the problem of distributed winner determination in combinatorial auctions is an important problem whose solution will enable the construction of sophisticated peer-to-peer marketplaces. It is also an interesting combination of distributed computation and distributed algorithmic mechanism design. Our algorithms and analysis are a first step towards the understanding of this problem and its ramifications.

References

1. Rothkopf, M.H., Pekec, A., Harstad, R.M.: Computationally manageable combinational auctions. Management Science **44** (1998) 1131–1147
2. Sandholm, T.: An algorithm for winner determination in combinatorial auctions. Artificial Intelligence **135** (2002) 1–54
3. Andersson, A., Tenhunen, M., Ygge, F.: Integer programming for combinatorial auction winner determination. In: Proceedings of the Fourth International Conference on MultiAgent Systems, IEEE (2000) 39–46
4. Nisan, N.: Bidding and allocation in combinatorial auctions. In: Proceedings of the ACM Conference on Electronic Commerce. (2000) 1–12
5. Zurel, E., Nisan, N.: An efficient approximate allocation algorithm for combinatorial auctions. In: Proceedings of the ACM Conference on Electronic Commerce. (2001)
6. Nisan, N., Ronen, A.: Algorithmic mechanism design. Games and Economic Behavior **35** (2001) 166–196
7. Feigenbaum, J., Shenker, S.: Distributed algorithmic mechanism design: Recent results and future directions. In: Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, ACM Press, New York (2002) 1–13
8. Hoos, H.H., Boutilier, C.: Solving combinatorial auctions using stochastic local search. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, AAAI Press / The MIT Press (2000) 22–29

9. Fujishima, Y., Leyton-Brown, K., Shoham, Y.: Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc. (1999) 548–553
10. Preist, C., Bartolini, C., Phillips, I.: Algorithm design for agents which participate in multiple simultaneous auctions. In: Agent-Mediated Electronic Commerce III, Current Issues in Agent-Based Electronic Commerce Systems (includes revised papers from AMEC 2000 Workshop), Springer-Verlag (2001) 139–154
11. Preist, C., Byde, A., Bartolini, C.: Economic dynamics of agents in multiple auctions. In: Proceedings of the fifth international conference on Autonomous agents, ACM Press (2001) 545–551
12. Wellman, M.P.: Market-oriented programming: Some early lessons. In Clearwater, S., ed.: Market-Based Control: A Paradigm for Distributed Resource Allocation. World Scientific (1996)
13. de Vries, S., Vohra, R.V.: Combinatorial auctions: A survey. INFORMS Journal on Computing **15** (2003) 284–309
14. Shehory, O., Kraus, S.: Methods for task allocation via agent coalition formation. Artificial Intelligence **101** (1998) 165–200
15. Willer, D., ed.: Network Exchange Theory. Praeger Publishers, Westport CT (1999)
16. Kakade, S.M., Kearns, M., Ortiz, L.E., Pemantle, R., Suri, S.: Economic properties of social networks. In Saul, L.K., Weiss, Y., Bottou, L., eds.: Advances in Neural Information Processing Systems 17. MIT Press, Cambridge, MA (2005)
17. Rosenschein, J.S., Zlotkin, G.: Rules of Encounter. The MIT Press, Cambridge, MA (1994)
18. Leyton-Brown, K., Pearson, M., Shoham, Y.: Towards a universal test suite for combinatorial auction algorithms. In: Proceedings of the 2nd ACM conference on Electronic commerce, ACM Press (2000) 66–76 `http://cats.stanford.edu`.
19. Wilensky, U.: NetLogo: Center for connected learning and computer-based modeling, Northwestern University. Evanston, IL (1999) `http://ccl.northwestern.edu/netlogo/`.