Bidtree ordering in IDA* combinatorial auction winner-determination with side constraints*

John Collins, Güleser Demir, and Maria Gini

Department of Computer Science and Engineering,
University of Minnesota
jcollins, gdemir, gini@cs.umn.edu

Abstract. We extend Sandholm's bidtree-based IDA* algorithm for combinatorial auction winner determination to deal with negotiation over tasks with precedence constraints. We evaluate its performance, and show that the *order* of items in the bidtree has a major impact on performance. Specifically, performance is enhanced if the items with the largest numbers of bids are at the top of the bidtree. This is due to the fact that the effective branching factor in the search tree is controlled by the number of bids returned from a query to the bidtree, which in turn is strongly related to its construction.

1 Introduction

The University of Minnesota's MAGNET (Multi-Agent Negotiation Testbed) system is an innovative agent-based approach to complex contracting and supply-chain management problems. The MAGNET system [5, 4] comprises a set of agents who negotiate with each other through a market infrastructure. It is designed to support the execution of complex plans among a heterogeneous population of self-interested agents. Such plans may involve complex task networks with temporal and precedence constraints.

Negotiation among agents in the MAGNET environment uses a combinatorial auction mechanism: A Customer agent may submit a Request for Quotes (RFQ) specifying a set of tasks, along with temporal and precedence constraints, and Supplier agents respond by submitting Bids. Each Bid specifies a set of tasks, a price, and resource availability data that includes task durations and early and late start limits. When the Customer agents receives those bids, it must solve an extended version of the Combinatorial Auction Winner Determination problem [1, 15] to find a minimum cost set of bids that covers all tasks and forms a feasible schedule.

Because the winner-determination problem must be solved in the context of a real-time negotiation scenario, and because the customer agent's RFQ must inform potential bidders of the earliest time they may expire their bids, we

* This work was supported in part by the National Science Foundation, awards NSF/IIS-0084202 and NSF/EIA-9986042

must be concerned with both performance and predictability of the winner-determination process.

The winner determination problem can be approached in various ways: dynamic programming [14], integer programming [1, 2], approximate solutions [6], and search methods [15, 16, 8]. The search methods have been shown to work very well on the average, scaling the winner determination problem up to hundreds of items and thousands of bids. In this work, we extend Sandholm's bidtree-based search approach [15] to deal with reverse-auction problems having precedence constraints among items, and show how to improve the search performance by controlling the order in which the bidtree, a central data structure in Sandholm's method, is generated.

The remainder of this paper is organized as follows: in Section 2 we describe our extended winner-determination problem; in Section 3, we describe our extensions to Sandholm's bidtree-based method, and we predict the impact of bidtree ordering on performance; in Section 4 we examine its performance, predictability, and scalability, and show how bidtree order affects performance; in Section 5 we summarize related work; and in Section 6 we conclude and suggest some loose ends that need further research.

2 Problem Description

MAGNET is designed to give an agent the ability to use market mechanisms (auctions, catalogs, timetables, supplier registries, etc.) to discover and commit resources needed to achieve its goals. We assume that agents are heterogeneous, self-interested, bounded-rational, and may be acting on behalf of different individuals or commercial entities who have different goals and different notions of utility. A core interaction in a MAGNET market environment is a first-price, sealed bid combinatorial reverse auction in which a Customer agent formulates one or more Requests for Quotations (RFQs) that describe its plans in the form of a task network with temporal and precedence constraints, and Suppliers respond by submitting bids specifying their willingness to perform subsets of the Customer's plan for specified costs and at specified times. Throughout this paper, we use the term "task" in place of the more traditional term "item" to describe the individual elements of the customer's RFQ.

In a MAGNET auction, each bid represents an offer to execute some subset of the tasks $\mathcal{S}_r = \{s_1, \cdots, s_m\}$ specified in the RFQ r (which we will frequently shorten to just $\mathcal{S}$), for a specified price, within specified time windows. Formally, a bid $b = (r, \mathcal{S}_b, \mathcal{W}_b, c_b)$ consists of a subset $\mathcal{S}_b \subseteq \mathcal{S}_r$ of the tasks specified in the corresponding RFQ r, a set of time windows $\mathcal{W}_b$, and an overall cost c_b. Each time window $w_s \in \mathcal{W}_b$ consists of an earliest start time $t_{es}(s, b)$, a latest finish time $t_{lf}(s, b)$, and a task duration $d(s, b)$.

It is a requirement of the protocol that the time window parameters in a bid b are within the time windows specified in the RFQ, or $t_{es}(s, b) \geq t_{es}(s, r)$ and $t_{lf}(s, b) \leq t_{lf}(s, r)$ for a given task s and RFQ r. This requirement may be relaxed, although it is not clear why a supplier agent would want to expose

resource availability information beyond that required to respond to a particular bid.

It is not required that the time windows specified in the RFQ satisfy the precedence constraints among tasks; what is required is that the time window specifications in the accepted bids must compose a feasible schedule. This allows customer agents to offer some additional flexibility to suppliers who may have existing resource commitments, thereby potentially lowering costs and increasing the number of bids, at the cost of receiving some bid combinations that cannot be used because they cannot be combined into a feasible schedule. As a result, the winner-determination search must be able to reject bid combinations whose time window specifications violate the precedence constraints among tasks. These constraints can be dealt with in either of two ways:

1. We can run a preprocessing step (described in [2]) that discovers *exclusion sets* of bids of size $[2 \ldots n]$, where n is the maximum depth of the precedence network. For each exclusion set of size n, at most $n - 1$ of its bids may be included in a solution.
2. We can discover constraint violations within the search process, during node evaluation. We can then simply discard the nodes that violate constraints.

Since the preprocessing step scales exponentially in the depth of the precedence network, we have chosen to deal with constraints using the second method, within the search.

We define a *partition* of the bids $\mathcal{P} \subseteq \mathcal{B}$ to be a set of bids such that each task $s \in \mathcal{S}$ is covered by exactly one bid $b \in \mathcal{P}$. The cost of a partition $\mathcal{P}$ is simply the sum of the costs of the bids in $\mathcal{P}$. A solution to the MAGNET winner-determination problem for m tasks $\mathcal{S} = \{s_1, \cdots, s_m\}$ and n bids $\mathcal{B} = \{b_1, \cdots, b_n\}$ is defined as the minimum-cost partition $\mathcal{P}_{\min}$that is consistent with the temporal and precedence constraints on the tasks as expressed in the RFQ and the mapped bids.

Figure 1 shows a very small example of the problem the winner-determination algorithm must solve. There is scant availability of carpentry resources, so we have provided an ample time window for that activity. At the same time, we have allowed some overlap between the Carpentry and Roofing tasks, perhaps because we believed this would attract a larger number of bidders with a wide variation in lead times and lower prices. Bid 2 indicates this carpenter could start as early as the beginning of week 3, would take 3 days, and needs to finish by the end of week 3. The lowest-cost bid for roofing is Bid 3, but we clearly can't use that bid with Bid 2. The lowest-cost complete, feasible combination for these three tasks is Bids 1, 2, and 5.

The winner determination problem for combinatorial auctions has been shown to be $\mathcal{NP}$-complete and inapproximable [15]. This result clearly applies to the MAGNET winner determination problem, since we simply apply an additional set of (temporal) constraints to the basic combinatorial auction problem, and we don't allow free disposal. In fact, because the additional constraints create additional bid-to-bid dependencies, and because bids can vary in both price and in time-window specifications, the bid-domination and partitioning methods used

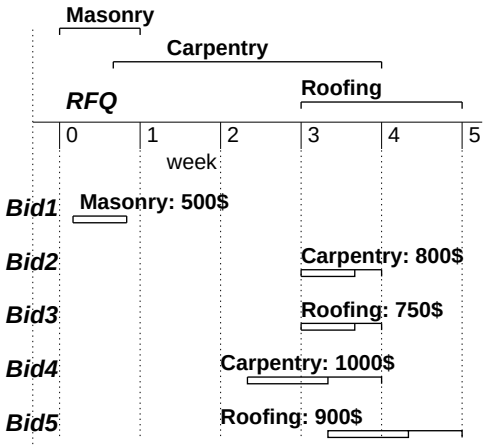

Fig. 1. Bid Example

by Sandholm to simplify the problem [15] cannot be applied to the MAGNET problem.

3 Bidtrees

In general, search methods are useful when the problem at hand can be characterized by a solution path in a tree that starts at an initial node (root) and progresses through a series of expansions to a final node that meets the solution criteria. Each expansion generates successors (children) of some existing node, expansions continuing until a final node is found. The questions of which node is chosen for expansion, and how the search tree is represented, lead to many different search methods. In the A* method, the node chosen for expansion is the one with the "best" evaluation[1], and the search tree is typically kept in memory in the form of a sorted queue. A* uses an evaluation function $f(N) = g(N) + h(N)$ for a node N, where $g(N)$ is an estimate of the cost of the path from initial node N_0 to node N, and $h(N)$ is an estimate of the remaining cost to a solution node. If $h(N)$ is a strict lower bound on the remaining cost (upper bound for a maximization problem), then A* is guaranteed to find a solution with the lowest evaluation, if any solutions exist. Iterative Deepening A* (IDA*) is a variant of A* that uses the same two functions g and h in a depth-first search which keeps in memory only the current path from the root to a particular node. In each iteration of IDA*, search depth is limited by a threshold value on the evaluation function f. At the first iteration, this threshold is typically the estimate of the cost of the initial node. Then, at each new iteration, the threshold is raised (or lowered for a maximization problem) by some formula.

[1] lowest for a minimization problem, highest for a maximization problem.

In this study we use both A* and IDA* formulations for the MAGNET winner-determination problem. This is a reverse auction, so we are looking for a minimum-cost set of bids that satisfy the criteria described in Section 2. A key decision in designing a search algorithm is finding a method that generates nodes in a "reasonable" order, preferably without generating duplicates. This is the purpose of the *bidtree* introduced by Sandholm in [15].

A bidtree is a binary tree of depth $m + 1$, where m is the number of tasks $|\mathcal{S}_r|$in the RFQ. Each non-leaf node corresponds to some task s and a subset $\mathcal{B}' \subseteq \mathcal{B}$ of the bids. Each non-leaf node has at most two children: one child is the subtree that contains the subset of bids $\mathcal{B}'(s, \text{in}) \subseteq \mathcal{B}'$ that include s (the *in* branch), and the other is the subtree that contains the complementary subset $\mathcal{B}'(s, \text{out}) = \mathcal{B}' - \mathcal{B}'(s, \text{in})$ (the *out* branch). Leaf nodes contain all non-dominated bids that cover specified task sets. In Sandholm's formulation, leaf nodes contain individual bids, because without the additional constraints, there can be only one non-dominated bid for a particular set of items, the only distinction among bids being item set and price. Because the precedence constraints create dependencies among bids with different task sets, bid domination is a much more complex issue in the MAGNET problem domain, so we will use bid sets at the leaves and defer the bid-domination issue for now.

As an example, suppose that we have a set of tasks $\mathcal{S} = \{s_1, \cdots, s_4\}$, and we have received a set of bids $\mathcal{B} = \{b_1, \cdots, b_{10}\}$ with the following contents: $b_1 : \{s_1, s_2\}$, $b_2 : \{s_2, s_3\}$, $b_3 : \{s_1, s_4\}$, $b_4 : \{s_3, s_4\}$, $b_5 : \{s_2\}$, $b_6 : \{s_1, s_2, s_4\}$, $b_7 : \{s_4\}$, $b_8 : \{s_2, s_4\}$, $b_9 : \{s_1, s_2\}$, $b_{10} : \{s_2, s_4\}$. Figure 2 shows a bidtree for this problem constructed with "lexical" task order, with task s_1 as the root of the tree, task s_2 as the second tier, etc.

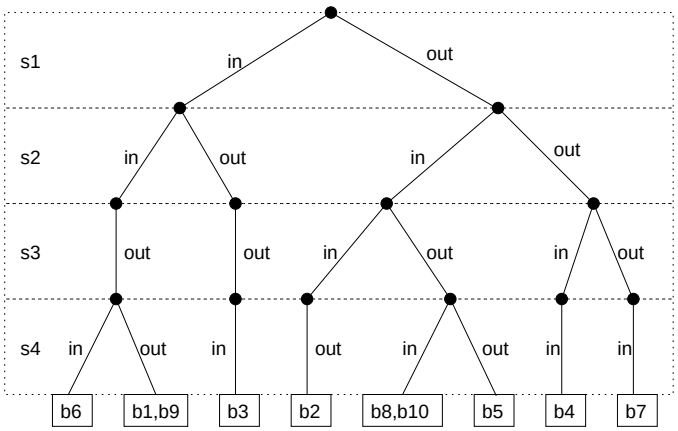

Fig. 2. Example bid tree, lexical task order

A bidtree supports lookup of bids based on task content, and determines the order in which bids are considered during the search. Moreover, it ensures that

each bid combination is tested at most once. We use the bidtree by querying it for sets of bids. A query consists of a "mask" vector, whose entries correspond to the "levels" in the bid tree. Each entry in the mask vector may take on one of the three values, $\{in, out, any\}$. A query is processed by walking the bid tree along with the mask vector. If an entry in the mask vector is *in*, then the *in* branch is taken at the corresponding level of the tree, similarly with *out*. If an entry is *any*, then both (*in* and *out*) branches are taken at the corresponding level of the bidtree. The result of a query is the set of "bid-buckets" containing the bids found at the leaf nodes that are encountered by processing the query (a bid-bucket is simply the set of bids in a single leaf node). For example, if we query the bidtree of Figure 2 with the mask $\{in, any, out, in\}$, the result will contain two bid-buckets $\{\{b_6\}, \{b_3\}\}$.

We now describe the process by which we use the bidtree to generate nodes in the search tree. A node N comprises a set of bids $\mathcal{B}_N$, an evaluation $f(N) = g(N) + h(N)$, and a set of "candidate bids" $\mathcal{B}_N^c$. The candidate bids for the root node are those bids returned by a query of the bidtree with a mask of $\{in, any, any, \cdots\}$, or all the bids that include the first task in the bidtree. The construction of a child of a parent node in a search tree is done using the bid tree as shown in detail in Figure 3. Note that there is no constraint-processing at this level. This is because the treatment of an infeasible node within the body of the A* or IDA* search is different depending on whether a node fails to expand (in which case its parent is discarded) or a particular child contains constraint violations (in which case the parent may yet have other feasible children).

There are several design decisions that must be made to implement the procedure shown in Figure 3. Among them is the order in which candidate bids are chosen, and the order in which tasks appear in the bidtree. We will explore the bidtree-order issue first.

Note that if a node N is a solution, then $\mathcal{B}_N$ is a partition; otherwise, it is a subpartition $\mathcal{P}_s(N)$. For every subpartition, there is a set of "complementary partitions" $\mathcal{P}_{\overline{s}}$ that can be combined with $\mathcal{P}_s(N)$ to form complete partitions $\mathcal{P}$. A subset of the complementary partitions is associated with each of the candidate bids for node N. Clearly, the "branching factor" for a node N (the number of child nodes N' in the search tree) is exactly equal to the number of candidate bids that do not lead to node abandonment in the procedure outlined above. Also, the mean branching factor in the subtree below N is a function of the size of the complementary partitions.

The order in which the tasks are represented in the bidtree can have a major influence on the sizes of the complementary partitions in the search tree. In Figure 4, we see bidtrees for the example problem of Figure 2, with the tasks sorted by increasing bid count (top), and by decreasing bid count (bottom). At first glance, they seem to be almost mirror images of each other.

The real difference between the two bidtrees in Figure 4 is the size of the complementary partitions in the nodes that are generated using them. This is easy to see with respect to the root node, where the set of candidate bids is the entire left subtree, and the sizes of the complementary partitions for the first

```
1  Procedure astar_expand
2  Inputs:
3      N: the node to be expanded
4  Output:
5      N′: a new node with exactly one additional bid, or null
6  Process:
7      buckets ← ∅
8      while buckets = ∅ do
9          if B^c_N = ∅ then return null
               "B^c_N is the set of candidate bids for node N"
10         b_x ← choose(B^c_N) "pick a bid from the set of candidates"
11         B^c_N ← B^c_N − b_x "remove the chosen bid from the set"
12         N′ ← new node
13         B_N′ ← B_N + b_x "B_N′ is the set of bids in node N′"
14         S_u ← unallocated_tasks(N′) "tasks not covered by any bid b ∈ B_N′"
15         mask ← create_mask(B′_N)
               "for each task that is covered by a bid in B_N′, set the corresponding
               entry to out. Then find the first task in s ∈ S_u (the task in S_u
               with the minimum index in the bidtree) and set its entry to in.
               Set the remaining entries to any"
16         buckets ← bidtree_query(B, mask)
17         B_u ← ∀s ∈ S_u, minimum_usable_bid(s) "see the narrative"
18         if (solution(N′))
               ∧((buckets ≠ ∅) ∨ (¬∃s|minimum_usable_bid(s) = null))
19         then
20             B^c_N′ ← buckets "candidates for N′"
21         else
22             remove(N, bucket(b_x))
                   "all bids in the bucket containing b_x in node N will produce the
                   same mask and therefore an empty candidate set or a task that
                   cannot be covered by any usable bid"
23     end while
24     g(N′) ← Σ_{b∈B_N′} c_b
25     h(N′) ← Σ_{b∈B_u} avg_cost(b)
26     return N′
```

Fig. 3. Bidtree-based node-expansion algorithm.

tier of children is at most the size of the right subtree. This indicates that the ideal task ordering is one in which the left subtree (the *in* branch) is always maximally larger than the right subtree (the *out* branch). This happens when the task with the largest *in* branch is at the top of the tree, followed by the task with the next largest *in* branch, etc., which corresponds to the ordering in Figure 4 (bottom). In Figure 5, we see the search trees that are generated by following our process using the two bidtrees shown in Figures 4.

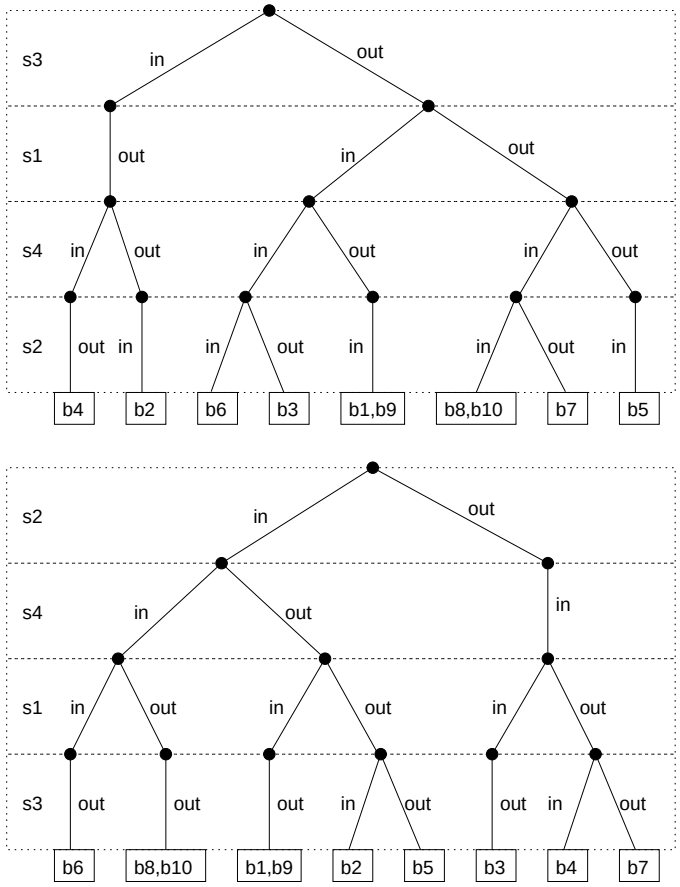

Fig. 4. Example bidtree sorted by increasing bid count (top) and by decreasing bid count (bottom).

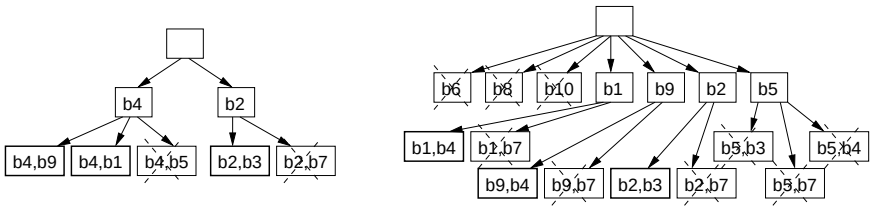

Fig. 5. Example search trees that result from using bidtrees with increasing bid count (left) and decreasing bid count(right). Incomplete nodes that can have no children are crossed out, and complete nodes are surrounded by dark boxes.

The number of children of a node is called the *branching factor*. We can place an upper bound on the branching factor as follows: Suppose we have m tasks

$\mathcal{S} = \{s_1, s_2, \cdots, s_m\}$ and n bids $\mathcal{B} = \{b_1, b_2, \cdots, b_n\}$, where the m tasks are included $i_1, i_2, \cdots, i_m$ times in the bids. Suppose $i_1 > i_2 > \cdots > i_m$. We term task b_1 the *most preferred* task since it has attracted the largest number of bids.

1. Suppose that the search tree is expanded starting with the most preferred task i_1. After expanding the root node using the bids containing s_1, then in the next level of the tree, each node contains a bid for s_1 and can have at most $n - i_1$ children. Obviously, this is a quite rough upper bound since each bid has certain subset of tasks, and for a particular node there might be no complementary partition at all. With similar reasoning, one expects at most $n - i_1 - i_2$, $n - i_1 - i_2 - i_3$, so on children for each node at the second, third etc. depths.
2. Alternatively, let us start the search tree with the least preferred task s_m. Then the subsequent second, third, fourth etc. depths of the search tree will have at most $n - i_m$, $n - i_m - i_{m-1}$, $n - i_m - i_{m-2}$ etc. children for each node.

A comparison of 1 and 2 reveals that at the same depth of the corresponding search tree, as $i_1 > i_2 > \cdots > i_m$ with n constant, the maximal branching factors for the decreasing bid count case is much less than the increasing bid count. therefore, as the numerical simulations confirm, the decreasing bid count gives much better performance.

The arguments above give us upper bounds, and they can obviously overestimate the children at a node. For a more realistic description of the relative branching factors, one has to deal also with the lower bounds on the children of a node. However, distributions of bids over the tasks as well as their interdependencies strongly affect the lower bound. In fact, as can be seen in Figure 7 below, the increasing sort does indeed occasionally win.

4 Performance

We wish to examine three issues experimentally: (a) the performance impact of bidtree order, (b) comparison of A* and IDA* with an Integer Programming formulation of the same problem, and (c) scalability of the IDA* formulation over task count and bid count. We have not attempted to reproduce Sandholm's original experiments, although that would certainly be desirable. That would require access to his tools, code, and problem instances, and we defer that to a future date.

Our experimental setup involves a plan generator, an RFQ generator, a bid generator, the solvers under test, and data collection and analysis tools. The plan generator produces random plans of specified sizes with a specified density of randomly-generated precedence relations. The RFQ generator uses expected-duration data, a specified amount of overall slack, and the CPM algorithm to generate time windows for the RFQs. The bid generator produces random bid sets by picking a starting task at random from the task network, attempting to

generate a bid for that task using random durations and offsets, and then following successor and predecessor relations with controlled probability to choose additional tasks. This is repeated recursively, resulting in bids that cover contiguous sets of tasks. Each experiment consists of 100 such randomly-generated problems.

The A* and IDA* algorithms are written in Java and integrated into the MAGNET customer agent framework. The IP solver is lp_solve[2], written in C. The preprocessor is written in Java and is also integrated with the agent framework, which invokes the solver as an external process. All experiments were run on a 1.8 GHz Pentium 4 machine with ample memory.

Because we are interested in the variability and predictability of the winner-determination process, we estimate and report the full distributions we observe in our experiments rather than mean values. We also show, for each experiment, the lognormal distribution that results from computing the Maximum Likelihood Estimators [10] on the sample data. Our general observation is that the lognormal distribution is an excellent estimate of the performance characteristics of the Integer Programming approach, while some of the A* and IDA* experiments exhibit a slightly heavier tail in their observed distributions.

In Figure 6 we see search-performance curves for the IP solver, and for A* and IDA* using a bidtree sorted by increasing bid count (labeled A*(inc) and IDA*(inc)), and for A* and IDA* using a bidtree sorted by decreasing bid count (labeled A*(dec) and IDA*(dec)). All five experiments used the same series of 100 randomly-generated problems. Clearly, the A* and IDA* perform much better, with lower variability, if the bid tree is constructed starting with the task having the largest number of bids[3].

Now we see graphically the impact of the lower branching factor induced by the decreasing sort of the bidtree. We can use an approximation known as the *effective branching factor* $b^\star$ to quantify the difference. If N is the total number of nodes generated and d is the depth of the solution found, then the effective branching factor $b^\star$ is the branching factor that a uniform tree of the depth d would have in order to contain N nodes. We can approximate this given that we know the depth of the branch of the tree that includes the solution. This is simply one more than the number of bids in the solution, since the root node contains no bids, and each expansion adds a bid to a partial solution. The value of $b^\star$ can be approximated as

$$\widetilde{b^\star} = N^{1/(d-1)}$$

For the two problem sets illustrated in Figure 6, the mean effective branch factors across the full set of 100 problems are

$$\widetilde{b^\star}_{\text{inc}} = 2.8$$

[2] available from ftp://ftp.ics.ele.tue.nl/pub/lp_solve

[3] In the interest of full disclosure, it should be noted that the A* search failed because of memory overflow (more than 30 000 nodes in the queue) for 6 of the 100 problems with the increasing sort, and for 1 of the 100 problems with the decreasing sort.

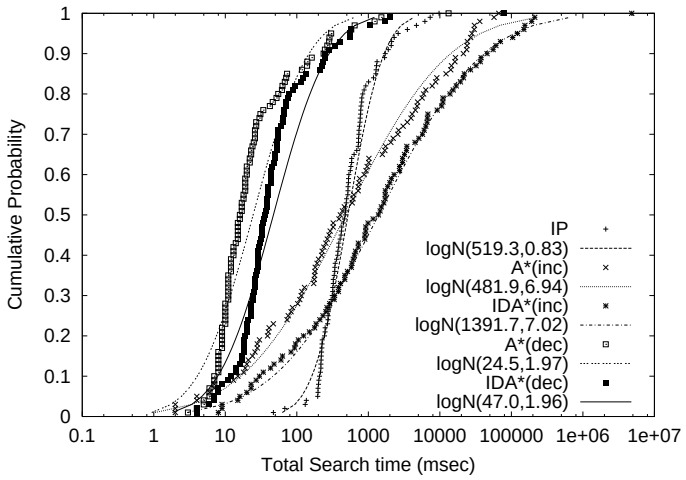

Fig. 6. Performance of A*, IDA*, and IP search (35 tasks, 123 bids) with bid tree sorted by increasing bid count (A*(inc) and IDA*(inc)), and by decreasing bid count(A*(dec) and IDA*(dec).

$$\widetilde{b^\star}_{\mathrm{dec}} = 1.7$$

where "inc" refers to the increasing bid-count sort, and "dec" refers to the decreasing bid-count sort.

Figure 7 is a scatter plot showing the relative performance of the IDA* search when using the two alternative bidtree orderings. For observations above the $x = y$ line, the decreasing sort performed better than the increasing sort. It clearly is not the case that the decreasing sort is *always* better. More interesting, perhaps, is the fact that there is very little correlation between the two orderings in terms of the difficulty of individual problems.

We also experimented with sorting the bid sets returned by a bidtree query. The idea is that if a large set is returned, then expanding with the lowest-cost bid first would generate the lowest-cost nodes earlier, thus reducing search effort. We could discern no impact on performance. Clearly, at least with the problems we are using, the performance improvement is no larger than the cost of sorting.

Finally, we show the beginnings of our explorations of scalability and predictability. Figure 8 shows results from a series of experiments in which the same series of 100 plans was used with bid sets ranging from 80 bids to 265 bids.

Because MAGNET agents must operate in a real-time negotiation environment, we are interested in being able to predict the time required to solve the winner determination problem in terms of estimates of problem size that might be available when composing the RFQ. Figure 9 shows an exponential regression curve fitted to the 95^{th} percentile points from the lognormal distributions shown in Figure 8.

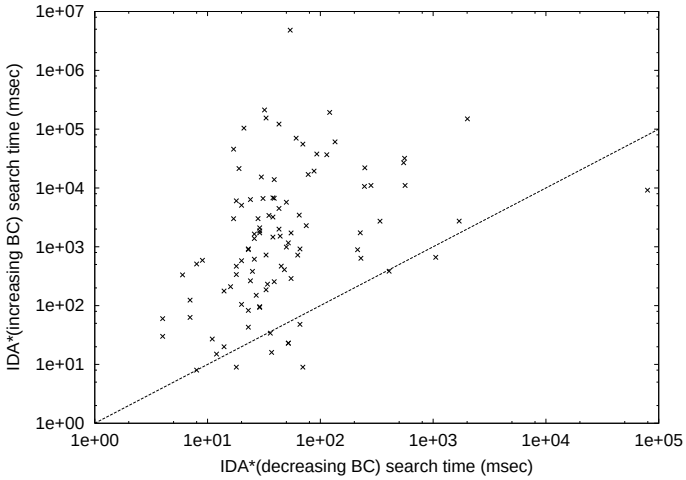

Fig. 7. Scatter plot showing relative performance of IDA* using increasing and decreasing sort on a set of 100 randomly-generated problems.

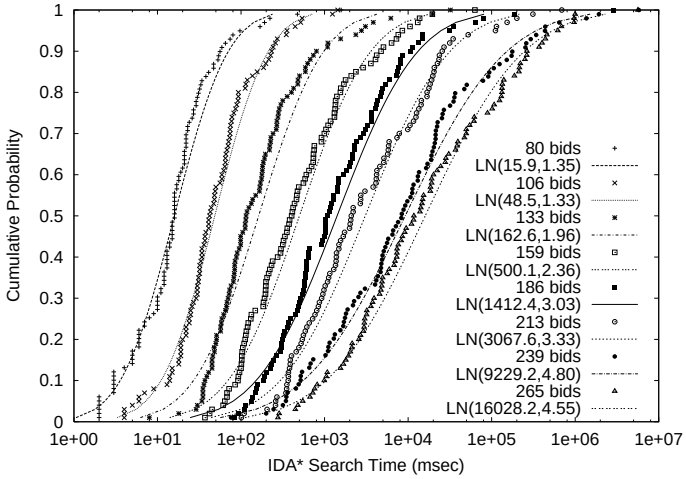

Fig. 8. IDA* run-time distributions for problems with 30 tasks, bid count ranging from 80 bids to 265 bids.

The goal of this type of analysis is to find empirical relationships between search effort and a variety of problem-size metrics that can be measured or estimated prior to submitting an RFQ. From Figure 9 we can see that at least one estimate of problem size can lead to good predictability. For additional experimental data and a more complete derivation of search-time prediction for the IP solver, see [3].

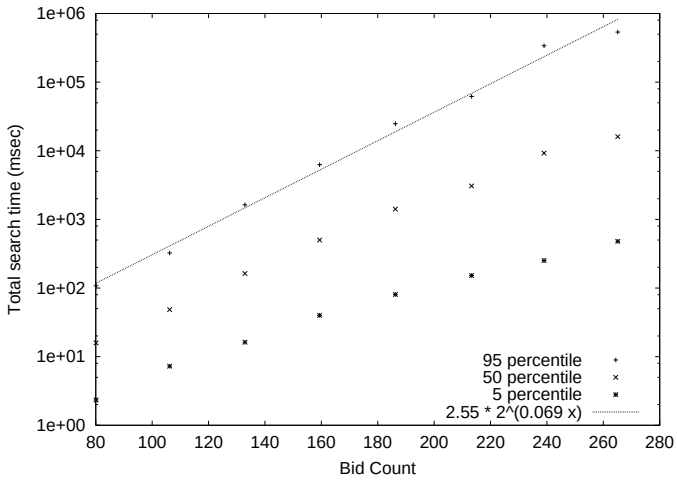

Fig. 9. 95^{th}, 50^{th} and 5^{th} percentile of runtime distribution from Figure 8, and regression curve that minimizes log square deviation.

5 Related Work

The combinatorial auction winner determination problem, which is equivalent to weighted set packing, is a well-known NP-hard problem [14], and it has been under dense investigation in recent years. It can be approached via dynamic programming [14] for small sets of bids. With certain restrictions on bids, Nisan [11] expresses the winner determination problem as a standard mixed integer programming problem, and formalizes several bidding languages. Andersson [1] also proposes an integer programming method showing that the integer programming enables the management of very general problems by the use of standard algorithms and commercially available software such as CPLEX. In the works of Rassenti [13] and Fujishima [6] the winner determination in combinatorial auctions is attempted via approximation algorithms. However, Sandholm [15] shows that no polynomial-time algorithm can be constructed to obtain a solution which is guaranteed to find a solution that is within any chosen ratio k of the optimum value. Search-based algorithms such as Bidtree [15], CASS [6] and CABOB [16] have proven very useful in large-scale problem since they are generally more scalable than the Integer Programming approach.

None of these methods deals with the additional constraints imposed on bids and bid combinations by the MAGNET domain. Several systems have attempted to organize task-oriented work among multiple agents. Parkes [12] describes an auction-based system for controlling access to shared railroad resources. It uses a mixed-integer approach, with many domain-specific optimizations. In [9] combinatorial auctions are used for the initial commitment decision problem, which is the problem an agent has to solve when deciding whether to join a proposed collaboration. Their tasks have precedence and hard temporal constraints. How-

ever, to reduce search effort, they use domain-specific *roles*, a shorthand notation for collections of tasks. In their formulation, each task type can be associated with only a single role. MAGNET agents are self-interested, and there are no limits to the types of tasks they can decide to do. In [7] scheduling decisions are made not by the agents, but instead by a central authority. The central authority has insight to the states and schedules of participating agents, and agents rely on the authority for supporting their decisions.

6 Conclusions and future work

We have discussed an extended version of the combinatorial auction winner determination problem to solve the bid evaluation problem in the MAGNET agent environment. In the MAGNET system, the agents participate in a combinatorial reverse auction over a set of tasks with precedence constraints. We show that a variation of the IDA* approach introduced by Sandholm is quite effective, and that the choice of bidtree ordering has a major impact on the performance of that algorithm because of its impact on the effective branching factor in the search tree.

Since the winner determination problem takes place in a real-time negotiation scenario, it needs to be solved in an acceptable and predictable time period. This requires that one must deal with both the performance and predictability of the winner determination process. As our results show, the performance of the search is considerably improved if the tasks with the largest number of bids are at the top of the bidtree. This results from the fact that the number of bids returned by a query influences the effective branching factor. In our case, building the bidtree with decreasing bid count order, one obtains a considerable reduction in the effective branching factor.

Several loose ends need to be filled in. We have not yet characterized search effort for the IDA* solver with respect to bid size and density of precedence relations in the task network, although much of this work has been done for the IP solver, see [3]. It is clear from comparing our results with those of Sandholm that the MAGNET problem is more difficult to solve than the standard combinatorial auction problem. It is also clear that the MAGNET problem approaches the standard problem as one reduces the complexity of the task network and the possibility of bib-to-bid temporal infeasibilities. There may be a threshold beyond which the additional preprocessing steps described by Sandholm become feasible and worthwhile. It is also important to understand the impact of bidtree ordering on combinatorial auctions that do not include side constraints. The arguments about partition size still hold, but the impact may be reduced by preprocessing steps that discover bid dominance and disjoint subsets of bids.

For current information and a list of publications on the University of Minnesota's MAGNET project, please see `http://www.cs.umn.edu/magnet`.

References

1. Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In *Proc. of 4th Int'l Conf on Multi-Agent Systems*, pages 39–46, July 2000.
2. John Collins and Maria Gini. An integer programming formulation of the bid evaluation problem for coordinated tasks. In Brenda Dietrich and Rakesh V. Vohra, editors, *Mathematics of the Internet: E-Auction and Markets*, volume 127 of *IMA Volumes in Mathematics and its Applications*, pages 59–74. Springer-Verlag, New York, 2001.
3. John Collins, Maria Gini, and Bamshad Mobasher. Multi-agent negotiation using combinatorial auctions with precedence constraints. Technical Report 02-009, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, Minnesota, February 2002.
4. John Collins, Wolfgang Ketter, Maria Gini, and Bamshad Mobasher. A multi-agent negotiation testbed for contracting tasks with temporal and precedence constraints. *Int'l Journal of Electronic Commerce*, 2002.
5. John Collins, Ben Youngdahl, Scott Jamison, Bamshad Mobasher, and Maria Gini. A market architecture for multi-agent contracting. In *Proc. of the Second Int'l Conf. on Autonomous Agents*, pages 285–292, May 1998.
6. Yuzo Fujishjima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proc. of the 16th Joint Conf. on Artificial Intelligence*, 1999.
7. Alyssa Glass and Barbara J. Grosz. Socially conscious decision-making. In *Proc. of the Fourth Int'l Conf. on Autonomous Agents*, pages 217–224, June 2000.
8. Holger H. Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In *Proc. of the Seventeen Nat'l Conf. on Artificial Intelligence*, pages 22–29, 2000.
9. Luke Hunsberger and Barbara J. Grosz. A combinatorial auction for collaborative planning. In *Proc. of 4th Int'l Conf on Multi-Agent Systems*, pages 151–158, Boston, MA, 2000. IEEE Computer Society Press.
10. Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, second edition, 1991.
11. Noam Nisan. Bidding and allocation in combinatorial auctions. In *1999 NWU Microeconomics Workshop*, 1999.
12. David C. Parkes and Lyle H. Ungar. An auction-based method for decentralized train scheduling. In *Proc. of the Fifth Int'l Conf. on Autonomous Agents*, pages 43–50, Montreal, Quebec, May 2001. ACM Press.
13. S. J. Rassenti, V. L. Smith, and R. L. Bulfin. A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics*, 13:402–417, 1982.
14. Michael H. Rothkopf, Alexander Pekeč, and Ronald M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.
15. Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.
16. Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *Proc. of the 17th Joint Conf. on Artificial Intelligence*, Seattle, WA, USA, August 2001.