AN INTEGER PROGRAMMING FORMULATION OF THE BID EVALUATION PROBLEM FOR COORDINATED TASKS

JOHN COLLINS AND MARIA GINI*

Abstract. We extend the IP models proposed by Nisan and Andersson for winner determination in combinatorial auctions, to the problem of evaluating bids for coordinated task sets. This requires relaxing the free disposal assumption, and encoding temporal constraints in the model. We present a basic model, along with an improved model that dramatically reduces the number of rows by preprocessing the temporal constraints into compatibility constraints. Experimental results show how the models perform and scale, and how they compare with a stochastic solver based on Simulated Annealing. On average, the IP approach finds optimum solutions significantly faster than Simulated Annealing, but with an extreme level of variability that may make it impractical in time-constrained agent negotiation scenarios.

Key words. automated negotiation, multiattribute combinatorial auctions, Integer Programming.

AMS(MOS) subject classifications. 90C10, 68T20

1. Introduction.

Business-to-business e-commerce is expanding rapidly, letting companies both broaden their customer base and increase their pool of potential suppliers. Negotiating supplier contracts for the multiple components that often make up a single product is complicated because time dependencies introduce a significant scheduling risk. Current e-commerce systems typically rely on either fixed-price catalogues or auctions [9], and they often don't deal effectively with the time dimension.

The University of Minnesota MAGNET (Multi-Agent Negotiation testbed) [3] system is designed to support the negotiation of contracts for coordinated tasks among a population of independent and self-interested agents. Dealing with coordinated tasks adds some major complexities to the auction model of agent interaction. First, because tasks have a temporal duration and precedence constraints, any bid selection algorithm must insure the temporal feasibility of the bids accepted. Second, because each of the tasks in the set of coordinated tasks is necessary to achieve the overall goal, any bid selection algorithm must insure complete coverage of the tasks. Third, delays and failures that might occur during the execution of the contracted tasks can threaten the accomplishment of the overall goal. This introduces the need to assess scheduling risk and to account for risk in the bid selection algorithm.

Several papers have been published recently that deal with linear and integer programming formulations of the allocation problem in combinato-

*Department of Computer Science and Engineering, University of Minnesota, Minneapolis, Minnesota. {jcollins,gini}@cs.umn.edu. Work supported in part by the National Science Foundation, awards NSF/IIS-0084202 and NSF/EIA-9986042.

rial auctions [1, 10]. This leads the the obvious question: can the MAGNET bid-evaluation problem be formulated as a linear or integer programming problem. If so, there are well-known and reasonably efficient evaluation procedures for these problems, although in general the integer programming problem is NP-complete. We show that an integer programming approach appears to be a feasible approach to the MAGNET bid-evaluation problem, and we evaluate its performance and suitability for a time-constrained agent reasoning process.

In the next section, we give a simple example that will help understand the details of our approach. Section 3 presents a straightforward IP model of the MAGNET bid-evaluation problem, and follows up with an improved version that uses preprocessing to generate a more compact representation. In Section 4, we present experimental results comparing the two IP formulations and a heuristic search method based on Simulated Annealing, and we show why the Simulated Annealing approach might be valuable, even though its average performance is much worse than the improved IP formulation. Finally, Section 5 relates this work to earlier work in solving combinatorial auction problems, and Section 6 presents our conclusions and suggestions for further work in this area.

2. Example.

Suppose we have a job to do that involves performance of a set of coordinated tasks within a limited time frame, and we wish to minimize the cost of the job. Examples might include constructing a building or a bridge, evacuating an island [12], shipping a large piece of industrial equipment overseas, or establishing a multi-link communication channel. The resources needed to perform those tasks must be acquired from self-interested suppliers, who are attempting to maximize the value of the resources under their control. It is the job of a MAGNET Customer agent to use an auction process to obtain a set of commitments for those resources, that can be composed into a temporally feasible plan, at a minimum price.

Figure 1 shows an example task network. It is a directed acyclic graph, with arcs representing precedence relations. (In this paper, the notion of "precedence" means that if task A precedes task B, written as $A \prec B$, then task A must be completed before task B can start. No delay is implied between the end of task A and the start of task B.) The numbers in parentheses are the expected durations of each task. We expect task duration data, as well as data on duration variability, resource availability, and supplier reliability, to be collected and made available from the MAGNET market infrastructure [3].

Suppliers submit bids on sets of tasks based on their own resource availabilities and costs, and based on a Request for Quotes (RFQ) submitted by a potential customer. A bid includes a set of tasks and a price, along with timing data, including duration and the earliest and latest times the task(s) may be started. When composing the RFQ for a plan, the cus-

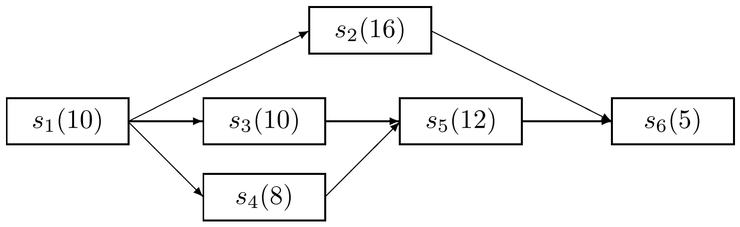

FIG. 1. *Example task network*

tomer must give suppliers some guidance about when the work must be done. This is done by specifying a *time window* for each task that gives the earliest start time and latest finish time. In generating this schedule, the customer has two conflicting goals:

1. Ensure that the bids received can be composed into a feasible plan. This can be done by specifying time windows that do not overlap.
2. Specify relatively wide, and possibly overlapping time windows, in hopes of attracting more bids and lower prices. The risk in doing this is that many bid combinations will not compose feasibly because their task time windows will be in conflict.

If we take the first approach, then the bid-evaluation process reduces to a variation on the combinatorial-auction winner-determination problem [16], without the free-disposal assumption [1]. The second approach leads to a much more difficult and interesting bid-evaluation problem, which is the subject of this paper.

The plan in Figure 1 has a *makespan* of 37 time units. This is the longest path through the graph. If we have a deadline for plan completion of 40 units, then we have an *overall slack* of 3 units, or about 8%.

TABLE 1
Example Bids

Bid	Task	Price	Early Start	Late Start	Duration
b_1	s_1	200	1.5	3.0	11.0
	s_3		12.5	14.0	8.5
	s_5		22.0	23.0	13.0
b_2	s_2	290	10.0	16.0	18.0
	s_6		28.0	33.0	6.0
b_3	s_4	160	9.0	13.0	6.0
	s_5		15.0	19.0	14.0
b_4	s_4	20	10.0	15.0	11.0

Table 1 shows some bids that might be received for this plan. Several

[1]The free-disposal assumption states that items can remain unallocated without penalty.

observations are apparent:

- Each bid may specify a “bundle” of tasks. A single price is given for the bundle.
- In this example and in the remainder of this paper, we assume that bids give early start, late start, and duration data for each task individually. This assumption can be relaxed.
- Bids b_1 and b_3 cannot both be accepted, because they both specify task s_5. Each task must be allocated to exactly one bid.
- Bids b_1 and b_2 cannot both be accepted, because the precedence relation $s_5 \prec s_6$ would be violated. This because the earliest time task s_5 could be completed under bid b_1 is 35, while the latest time task s_6 could start under bid b_2 is 33.
- Bids b_1 and b_4 cannot both be accepted. This is more subtle. The precedence relations $s_1 \prec s_4$ and $s_4 \prec s_5$ can be satisfied individually. However, when we attempt to combine the two bids, we see that, in order to satisfy $s_1 \prec s_4$, the early finish time of task s_4 is pushed back to 23.5, and $s_4 \prec s_5$ is violated.

We'll use this example to examine details of our bid-evaluation process.

3. IP formulations.

We start by introducing some notation. A plan consists of a set $\mathcal{S}$ of tasks $s_j, j = 1..m$. Each task s_j has a precedence set $\mathcal{P}_j = \{s_{j'} | s_{j'} \prec s_j\}$, the set of tasks $s_{j'}$ that must be completed before s_j is started. At the conclusion of some bidding process, we have a set $\mathcal{B}$ of bids $b_i, i = 1..n$. Each bid b_i specifies a set of tasks $\mathcal{S}_i$ and a price p_i. For each task s_j, a bid b_i that includes the task ($s_j \in \mathcal{S}_i$) may specify an early start time e^i_j, a late start time f^i_j, and a duration d^i_j.

We have also defined a risk factor r_i, associated with each bid, that is based on static factors, such as the reputation of the bidder. The derivation of r_i is outside the scope of this discussion, we include it for completeness.

3.1. A straightforward model.

First, we present a “direct” model, in which the coverage and feasibility constraints are directly represented. We associate a 0/1 variable x_i with each bid b_i, with the sense that in a solution where $x_i = 1$, b_i is accepted. No pre-processing is required other than composing the constraint rows. If we include only static risk factors (those that are determined strictly by the set of bids chosen, and not by scheduling considerations), the formulation of the basic bid-evaluation problem is:

Minimize:

$$\sum_{i=1}^{n} (p_i + r_i) x_i$$

Subject to:

- Bid selection – each bid is either selected or not selected. These are the integer variables that make this an integer programming problem.

$$x_i \in \{0,1\}$$

- Coverage – each task s_j must be included exactly once.

$$\forall j = 1..m \sum_{i|s_j \in \mathcal{S}_i} x_i = 1$$

Note that under the free disposal assumption, each task would be included at most once, rather than exactly once.

- Local feasibility – each task s_j must be able to start after the earliest possible completion time of each of its predecessors $s_{j'}$. This constraint ignores global feasibility. In other words, here we are looking only at the start times of a particular task and its immediate predecessors.

$$\begin{aligned}&\forall j = 1..m, \forall i|s_j \in \mathcal{S}_i, \forall i'|s_{j'} \in (\mathcal{S}_{i'} \cap \mathcal{P}_j),\\ &x_i f_j^i \geq x_{i'}(e_{j'}^{i'} + d_{j'}^{i'}) - M(1 - x_i)\end{aligned}$$

where M is a "large" number (we typically use a value of 10^{12}), and the last term $M(1-x_i)$ is used to make the constraint satisfied in the case where $x_i = 0$.

In our example, the precedence relations between b_1 and b_4 would be expressed as

$$\begin{aligned}x_4(15.0) &\geq x_1(1.5 + 11.0) - M(1 - x_4)\\ x_1(23.0) &\geq x_4(10.0 + 11.0) - M(1 - x_1)\end{aligned}$$

- Global feasibility – each task must be able to start after the earliest possible completion time of each of its predecessors, where the predecessors may in turn be constrained not by their bids, but by their respective predecessors.

$$\begin{aligned}&\forall j = 1..m, \forall i|s_j \in \mathcal{S}_i, \forall i'|s_{j'} \in (\mathcal{S}_{i'} \cap \mathcal{P}_j), \forall i''|s_{j''} \in (\mathcal{S}_{i''} \cap \mathcal{P}_{j'}),\\ &\quad x_i f_j^i \geq x_{i'} d_{j'}^{i'} + x_{i''}(e_{j''}^{i''} + d_{j''}^{i''}) - M(1 - x_i)\\ &\forall j = 1..m, \forall i|s_j \in \mathcal{S}_i, \forall i'|s_{j'} \in (\mathcal{S}_{i'} \cap \mathcal{P}_j),\\ &\quad \forall i''|s_{j''} \in (\mathcal{S}_{i''} \cap \mathcal{P}_{j'}), \forall i'''|s_{j'''} \in (\mathcal{S}_{i'''} \cap \mathcal{P}_{j''}),\\ &\quad x_i f_j^i \geq x_{i'} d_{j'}^{i'} + x_{i''} d_{j''}^{i''} + x_{i'''}(e_{j'''}^{i'''} + d_{j'''}^{i'''}) - M(1 - x_i)\\ &\quad \vdots\end{aligned}$$

In our example, the infeasibility between b_1 and b_4 is captured by the constraint

$$x_1(23.0) \geq x_4(11.0) + x_1(1.5 + 11.0) - M(1 - x_1)$$

The number of constraints generated by these formulas is highly variable, depending strongly on the details of the submitted bids and how they interact with the precedence network in the plan. For example, in the 5 tasks – 20 bids case described in Section 4, the count varied from 10 to 24000.

3.2. Collapsing the feasibility constraints. The formulation given in Section 3.1 is correct, but it can be dramatically improved, based on several observations. The first is that we can pre-process the coverage constraints to reduce the number of bids. If there is any task s_j for which only one bid b_i has been received (we'll call bid b_i a "singleton" bid for task s_j), b_i must be part of any complete solution. Bids b_k that conflict with b_i can then be discarded. In more formal terms,

$$\forall j | \sum_{i|s_j \in \mathcal{S}_i} 1 = 1, x_i = 1, \forall k | \mathcal{S}_i \cap \mathcal{S}_k \neq \emptyset, x_k = 0$$

This test is repeated until no further singleton bids are detected.

Next, we make the following observations regarding the feasibility constraints:

1. We have generated feasibility constraints between bids that cannot possibly be part of the same solution, because they contain overlapping task sets. The coverage constraints will ensure that only one bid will be chosen to cover each task. We can discard such constraints immediately. In our example, this means that we need not generate or evaluate any feasibility constraints that include both b_1 and b_3.
2. The feasibility constraints can be greatly simplified by doing the arithmetic during preprocessing, and including only those constraints that can have an impact on the outcome. This way, we eliminate all the feasibility constraints shown in Section 3.1, and replace them with a much smaller number of simple compatibility constraints. In other words, if we start with a constraint of the form

$$x_i f_j^i \geq x_{i'}(e_{j'}^{i'} + d_{j'}^{i'}) - M(1 - x_i),$$

   we compute $f_j^i - (e_{j'}^{i'} + d_{j'}^{i'})$. Just in case the result is negative, we include a constraint of the form

$$x_i + x_{i'} \leq 1$$

   which will prevent both x_i and $x_{i'}$ from being part of the same solution. This simplification can be similarly applied to the global feasibility constraints. In general, such constraints tell us that for some combination of n bids, at most $n - 1$ of them may be part of a solution.

If either x_i or $x_{i'}$ in the above formula is a singleton, then clearly the other cannot be part of a solution, so it can be eliminated. Also, if both x_i and $x_{i'}$ are singletons, then we know the problem cannot be solved.

In our example, we can observe the infeasibility between b_1 and b_4 during pre-processing, and rather than generate the formula given above in Section 3.1, we replace it with

$$x_1 + x_4 \leq 1$$

3. If we have successive tasks in the same bid, we can filter the bids themselves for internal feasibility prior to evaluation. After the previous step, any constraints of the form $x_i + x_i \leq 1$ represent bids x_i that can be discarded.

3.3. Minimizing completion time. If we want to minimize the time to complete the plan, we must develop an expression for the completion time. To begin with, we define t_0 as the (fixed) start time of the plan. Then we need to determine the latest time at which some task will be completed. We needn't consider all tasks, just the "leaf tasks," those that have no successors. If we ignore precedence constraints, the earliest possible completion time t_c for the plan as a whole is the maximum early finish time of any leaf task for a given bid assignment. Since in any valid bid assignment, only one bid is chosen for any given task, we can express this as

$$t_c = \max_{j | \forall k, s_j \notin \mathcal{P}_k} \sum_{i | s_j \in \mathcal{S}_i} x_i (e^i_j + d^i_j)$$

where a task s_j that has no successors is one that is in the predecessor set of no other tasks.

Unfortunately, it doesn't work to ignore precedence constraints. There may be a task in the precedence set of any given leaf task s_j whose early finish time in a given bid assignment will prevent s_j from starting at its early start time. To avoid this problem, while taking advantage of the precedence constraints we've already developed, we do two things. First, we must have a single task whose completion marks the end of the plan; to ensure that this is the case, we create a dummy task s_c, with 0 duration. We then define t_c as the start time of task s_c. We don't define a bid for this task, because we want to use its start time as a variable. We also define the plan start time t_0. Then we have to add a completion-time term to the objective function, which now reads:

Minimize:

$$W_c \sum_{i=1}^{n} (p_i + r_i) x_i + W_t (t_c - t_0)$$

where W_c is the relative weight given to cost, W_t is the relative weight given to completion time, and $(t_c - t_0)$ is the total makespan of the plan.

Next, we add an additional set of feasibility constraints, as given above in Section 3.1, to constrain the dummy task to start later than the completion times of all the leaf tasks. This set will include the local and global feasibility constraints, expanded recursively to the root tasks, substituting t_c for $x_i f_j^i$. Since t_c is a variable that appears in the objective function, its final value with be the earliest time that is greater than the maximum early completion time over all leaf tasks for any given bid assignment.

This approach does not work with the simplified form of the feasibility constraints as given in Section 3.2. There, we have discarded the temporal information in preprocessing, and are left with simple compatibility constraints. This deprives the IP solver of the information necessary to operate on completion time. An alternative approach in this case wold be to make completion time be a constraint, rather than a factor in the objective function. This would typically require multiple passes through the IP solver to find an acceptable combination of price and completion time.

4. Experimental results.

To illustrate the effectiveness of our formulation, we have implemented both the original formulation given in Section 3.1, and the pre-processed formulation given in Section 3.2, without the completion-time extension. Due to resource constraints, we were only able to run the original formulation on very small problems. We also ran our Simulated Annealing (SA) [13] search engine [4] on the same problems for comparison. In general, results show that our IP formulation is practical for moderate-sized problems, though it scales exponentially. On average, it performs better than the SA approach (as it was tuned for these experiments) for smaller problems. On the other hand, the time required to find the optimum solution exhibits a very high variability, while SA is an anytime method that can usually find "good" solutions in a much more predictable amount of time. Experiments were run using dual-processor 850 MHz Linux boxes, and the Sun Java HotSpot compiler in client mode. Timings are given in wall-clock time.

The MAGNET system is written in Java, and the IP solver is lp_solve, available from ftp://ftp.ics.ele.tue.nl/pub/lp_solve/. Because we are using an out-of-process IP solver, some well-known techniques, such as starting with a subset of the constraint set and adding additional constraints only if they are violated, or recording multiple solutions as the solver runs, are not possible in our current environment.

Each problem set consists of 200 problems, with randomly-generated plans and randomly-generated bids. Our problem generator has a large number of parameters; we kept all of them constant except for the task-count, bid-count, and bid-size values. Since our goal is to evaluate bids in a time-limited multi-agent interaction situation, we are primarily interested in scalability and predictability. Secondarily, we are interested in discovering measurable problem characteristics that the agent can use to tune its evaluation process "on the fly."

The process for generating problems operates as follows:

1. *Generate plan*: The desired number of tasks is generated, and random precedence relations are created between them, avoiding redundant precedence links. Tasks are randomly selected from among three "task types" that specify different values of expected duration, duration variability, and expected resource availability.
2. *Compose RFQ*: An RFQ is generated by setting time windows for the tasks in the plan, and specifying the timeline for the bidding process. Time windows are set by determining the makespan of the plan (the longest path through the precedence network) and multiplying it by a "slack" factor of 1.2, then "relaxing" the time windows for individual tasks to allow some overlap. The final result is that individual tasks are given time windows of at least 125% of their expected values (tasks not on the critical path will have longer time windows).
3. *Generate Bids*: A specified number of attempts are made to generate bids against the RFQ. Each bid is generated by selecting a task at random from the plan, using the task-type parameters to generate a supplier time window for that task, and testing this time window against the time window specified in the RFQ for that task. If the supplier's time window is contained within the RFQ time window, we call it a "valid task spec" and add the task to the bid. If a valid task spec was generated, then with some probability, each predecessor and successor link from that task is followed to attempt to add additional tasks to the bid, and so on recursively. The resulting bids specify "contiguous" sets of tasks, and are guaranteed to be internally feasible. Finally, a cost is determined for the overall bid. Because valid task specs are not always achieved, some attempts to generate bids will fail altogether.

The first experiment compares the performance of the original IP formulation to the revised formulation. We were only able to run the original formulation on the smallest problems (5 tasks, up to 20 bids) because some problems were generating more than 10^5 rows and taking inordinate amounts of time to solve (we stopped one after 13 hours). Table 2 shows the results of this experiment. All entries are averaged across 200 problems. Key features to observe are the relative problem sizes (number of rows) and the extreme variability (given as σ(time)) of the original formulation. For the revised formulation, the reported time is the sum of preprocessing time and IP solver time. The time required for preprocessing is generally between 2 and 20 times the run time of the IP solver itself, but this appears to be time well spent. Our experience attempting to solve larger problems with the original formulation shows that the advantage of preprocessing grows dramatically as problem size increases.

Table 2
Comparing IP formulations

		Original IP			Revised IP		
Task Count	Bid Count	Rows	Time (msec)	σ(time)	Rows	Time (msec)	σ(time)
5	11.4	487	195	1001	16.3	71.3	144
5	13.5	869	233	937	18.6	71.0	150
5	15.0	1265	753	5125	20.3	68.8	125
5	17.0	2271	3150	16256	22.7	95.8	279

The second series demonstrates scalability of the search process, using both the revised IP formulation and our Simulated Annealing solver, as the size of the plan varies, with a (nearly) constant ratio of bids to tasks (the ratio varies somewhat due to the random nature of the bid-generation process). Tables 3 and 4 show problem characteristics for this set. In both tables, the "Solved" column gives the number of problems solved out of 200 (not all 200 problems were solvable). In Table 3 "Opt" shows the number of times the optimum solution was found, "Time" is the time taken and σ(time) gives the standard deviation of the "Time" column. It should be noted that the performance results for the SA search are somewhat arbitrary; the annealing schedule and stopping conditions can be adjusted at will with respect to any measurable characteristic of the problem. Clearly, the SA search would have achieved better optimization results on the larger problems if we scaled it more strongly with respect to problem size. On the other hand, the SA search does often find optimal solutions, even though there is no way to know they are optimal. The non-optimal solutions it reports are typically within a few percent of optimal, even though the typical feasible solution in this environment is about 50% worse than optimal.

Table 3
Task size experiment: Simulated Annealing

Task Count	Bid Count	Bid Size	Solved	Opt	Time (msec)	σ(time)
5	13.5	2.14	188	188	10	10
10	28.5	3.20	184	178	147	391
15	45.4	4.35	179	141	524	986
20	60.8	5.21	169	118	1171	1707
25	77.6	6.30	147	89	1958	2683
30	93.3	7.41	141	80	3445	4012
35	110.6	8.69	116	66	4012	4289

In Table 4, the times for preprocessing and for the IP solver are given separately, as "PP" and "IP". The σ(time) column gives the standard deviation of the sum of these two times. The individual variabilities of the PP time and the IP time are comparable – the large values of σ in the table are not primarily due to either process.

TABLE 4
Task size experiment: Revised IP

Task Count	Bid Count	Bid Size	Solved	Rows	PP (msec)	IP (msec)	σ(time)
5	13.5	2.14	188	19.1	10.4	34.5	15
10	28.5	3.20	184	44.3	70.1	34.3	57
15	45.4	4.35	181	85.6	191	30.2	170
20	60.8	5.21	176	145	489	40.7	380
25	77.6	6.30	165	281	1268	56.2	1423
30	93.3	7.41	157	514	3375	194	4353
35	110.6	8.69	137	861	5181	317	7100

Figure 2 shows graphically the relative average performance of the SA and IP approaches on the task-size experiment. The striking feature of this graph is the apparent improvement in performance of the SA approach as problem size increases. This is an artifact of the tuning parameters, which are cutting off the search too soon on the larger problems. We can clearly see the corresponding dropoff in optimization performance in Table 3.

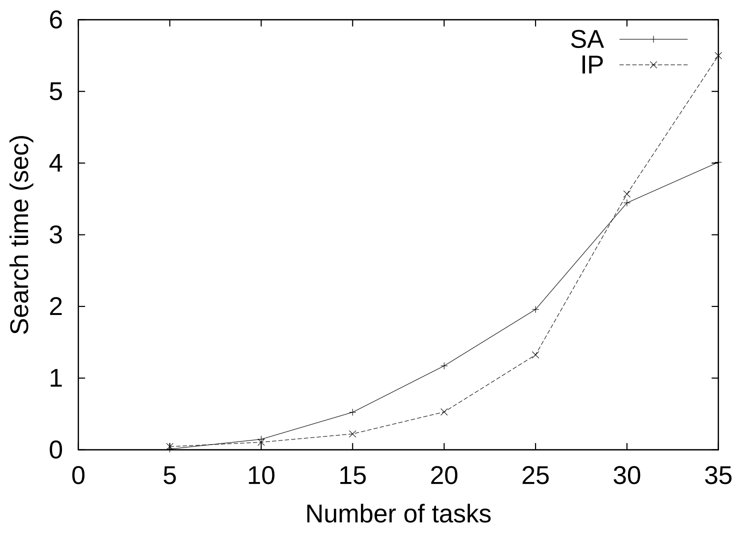

FIG. 2. *Search time as a function of task count*

In Figure 3, we show the result of varying the bid count with a fixed plan size of 20 tasks. Here, the exponential tendency of the IP method is clear, while the time required by the SA method is close to linear. This is accompanied by a corresponding fall-off of optimization performance of the SA method as the density of solutions increases. We see this in the data set labeled “SA opt”, which plots the ratio of the optimum solution as determined by the IP search, to the solution reported by the SA method.

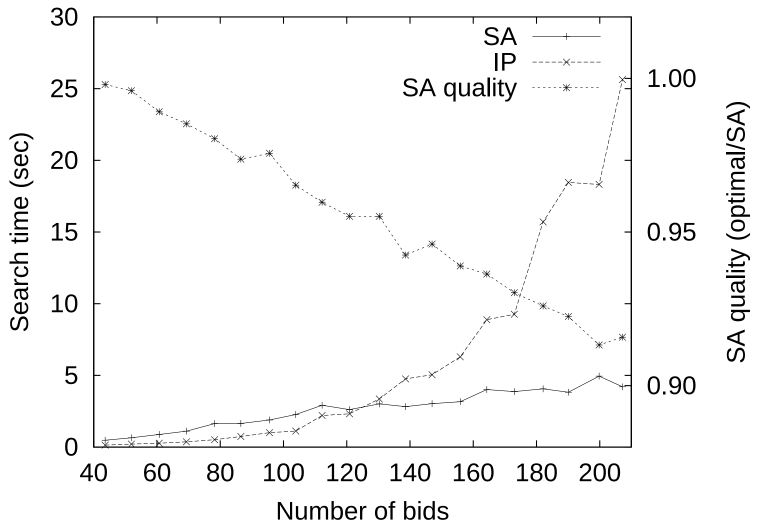

FIG. 3. *Search time as a function of bid count*

Finally, in Figure 4, we see the effect on search effort as the average size of bids is varied. For this set, the task count was held constant at 20 tasks, and the bid count was 70 bids. As in the previous set, the optimization performance of the SA method is traded off against run time to produce these results.

It is apparent from these experiments and others that the probability of finding the optimal solution in a given amount of time is higher with the IP approach than with the SA approach. For the purpose of supporting an agent involved in a negotiation process, the key difference between the SA and IP search methods seems to be in controllability and predictability. Both methods exhibit a high variability in the amount of time required to produce a solution, and no significant correlations between overall problem characteristics and the amount of time required have been found. Nor is there a strong correlation between the quality of the SA result and the time required for the IP solution. More significantly, perhaps, there is no correlation (correlation coefficient < 0.1 on all problem sets) between the IP solution time and the SA solution time even for problems in which SA found the optimal solution. For an agent that must make a decision in a

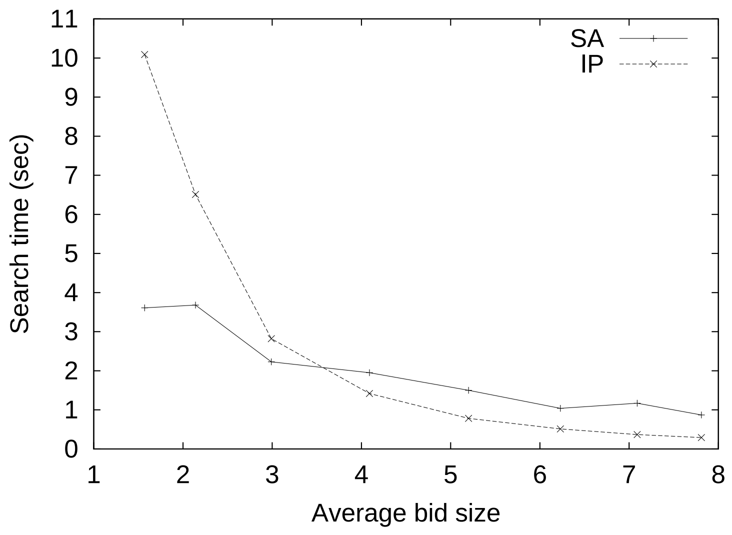

FIG. 4. *Search time as a function of bid size*

fixed amount of time, the extreme variability of either approach presents an unacceptable situation. It appears that an ideal approach would be to run both methods in parallel. Both the IP method and the SA method can be configured to deliver candidate solutions “on the fly”, and SA solutions can be used to bound the IP search. This can be especially significant in the cases where preprocessing time dominates the IP solution time.

The results given here for the Simulated Annealing search method do not compare well with the results reported in [4], because we have eliminated the notion of bid break-downs. In the original system, there was an assumption that suppliers who submitted bids on multiple tasks were also obligated to perform any subset of those tasks. This allowed the search engine to “take apart” the bids it was given, and greatly simplified the problem of finding a feasible, if suboptimal solution. Without the bid break-down assumption, the problem has become significantly more difficult to solve, and the version of the Simulated Annealing search reported here is tuned to anneal more slowly and to search as much as 10 times longer before terminating.

5. Related work. The determination of winners of combinatorial auctions [8] is known to be hard. Many algorithms have been proposed to produce good or optimal allocations. Dynamic programming [14] produces optimal allocations, but works well only for small sets of bids, and imposes significant restrictions on the class of bids. Nisan [10] formalizes several bidding languages and compares their expressive power. He analyzes different classes of auctions, and proposes an approach based on Linear Programming for bid allocation. Shoham [5] produces optimal allocations for OR-bids with dummy items by cleverly pruning the search space. Sandholm [15, 17] uses an anytime algorithm to produce optimal allocations for a more general class of bids, which includes XOR and OR-XOR bids. Andersson [1] proposes integer programming for winner determination in combinatorial auctions. The major difference is that in the cases studied for combinatorial auctions, bid allocation is determined solely by cost. Our setting is more general. Our agents have to cover all the tasks, ensure feasibility of the bids they accept, and reduce scheduling risk.

Walsh has proposed using combinatorial auction mechanisms for supply-chain formation [18] and for decentralized scheduling [19]. Neither of these proposals requires the allocation solver to deal with temporal feasibility, which is the principal problem dealt with in this work.

One of the algorithms we used is based on simulated annealing [13], and as such combines the advantages of heuristically guided search with some random search. Since the introduction of iterative sampling [7], a strategy that randomly explores different paths in a search tree, there have been numerous attempts to improve search performance by using randomization. A variety of methods that combine randomization with heuristics have been proposed, such as Least Discrepancy Search [6], heuristic-biased stochastic sampling [2], and stochastic procedures for generating feasible schedules [11], just to name a few.

6. Conclusion and future work. This work was motivated by the need for an improved bid evaluation procedure for the MAGNET system, and by the recent successes reported in applying Integer Programming to the related Combinatorial Auction winner determination problem. We have shown that Integer Programming can be applied to the problem of bid evaluation in a combinatorial auction situation that includes temporal constraints among items, and lacks the free-disposal option. The formulation we present typically requires an amount of pre-processing that is significantly greater than the time required by the IP solver itself. A significant weakness of the IP approach is the high degree of variability in run time. For an agent that must perform on a fixed time schedule, this may not be acceptable. We suggest that running a stochastic search in parallel with an IP solver may alleviate this drawback, and provide experimental data that supports this approach.

Several open questions remain. Many IP solution methods require a much tighter integration between the host application and the IP solver than we have been able to implement. This would allow the application to set bounds, add constraints, and record interim (potentially suboptimal) integer solutions. Such an integration would allow exploration of the potential synergy between a heuristic search engine and an IP solver in time-constrained situations such as a MAGNET agent must face. In addition, the data reported here make it clear that the Simulated Annealing search method needs to be better-tuned across the range of problem sizes. Both the stopping criteria and the annealing schedule need to be better adjusted in response to multiple problem-size measures. We have explored several problem-size measures in this work, but others might also be important, such as the degree of variability in bid size, task coverage, or cost/task.

It seems clear that further simplifications could be achieved in preprocessing. For example, when a singleton is encountered, it could tighten the time constraints, and the effects could be propagated through the task network. We would also like to understand how to incorporate risk factors that depend on the distribution of slack in the schedule, and how to optimize completion time without significantly increasing the complexity of the problem.

REFERENCES

[1] A. Andersson, M. Tenhunen, and F. Ygge, *Integer programming for combinatorial auction winner determination*, in Proc. of 4th Int'l Conf on Multi-Agent Systems, July 2000, pp. 39–46.

[2] J. L. Bresina, *Heuristic-biased stochastic sampling*, in Proc. of the Thirteenth Nat'l Conf. on Artificial Intelligence, 1996.

[3] J. Collins, C. Bilot, M. Gini, and B. Mobasher, *Mixed-initiative decision support in agent-based automated contracting*, in Proc. of the Fourth Int'l Conf. on Autonomous Agents, June 2000, pp. 247–254.

[4] J. Collins, R. Sundareswara, M. Gini, and B. Mobasher, *Bid selection strategies for multi-agent contracting in the presence of scheduling constraints*, in Agent Mediated Electronic Commerce II, A. Moukas, C. Sierra, and F. Ygge, eds., vol. LNAI1788, Springer-Verlag, 2000.

[5] Y. Fujishima, K. Leyton-Brown, and Y. Shoham, *Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches*, in Proc. of the 16th Joint Conf. on Artificial Intelligence, 1999, pp. 548–553.

[6] W. D. Harvey and M. L. Ginsberg, *Limited discrepancy search*, in Proc. of the 14th Joint Conf. on Artificial Intelligence, 1995, pp. 607–613.

[7] P. Langley, *Systematic and nonsystematic search strategies*, in Proc. Int'l Conf. on AI Planning Systems, College Park, Md, 1992, pp. 145–152.

[8] R. McAfee and P. J. McMillan, *Auctions and bidding*, Journal of Economic Literature, 25 (1987), pp. 699–738.

[9] P. Milgrom, *Auction and bidding: a primer*, Journal of Economic Perspectives, 3 (1989), pp. 3–22.

[10] N. Nisan, *Bidding and allocation in combinatorial auctions*, in Proc. of ACM Conf on Electronic Commerce (EC'00), Minneapolis, Minnesota, October 2000, ACM SIGecom, ACM Press, pp. 1–12.

[11] A. Oddi and S. F. Smith, *Stochastic procedures for generating feasible schedules*, in Proc. of the Fourteenth Nat'l Conf. on Artificial Intelligence, 1997, pp. 308–314.

[12] M. E. Pollack, *Planning in dynamic environments: The DIPART system*, in Advanced Planning Technology, A. Tate, ed., AAAI Press, 1996.

[13] C. R. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley & Sons, New York, NY, 1993.

[14] M. H. Rothkopf, A. Pekeč, and R. M. Harstad, *Computationally manageable combinatorial auctions*, Management Science, 44 (1998), pp. 1131–1147.

[15] T. Sandholm, *An algorithm for winner determination in combinatorial auctions*, in Proc. of the 16th Joint Conf. on Artificial Intelligence, 1999, pp. 524–547.

[16] ———, *Approaches to winner determination in combinatorial auctions*, Decision Support Systems, 28 (2000), pp. 165–176.

[17] T. Sandholm and S. Suri, *Improved algorithms for optimal winner determination in combinatorial auctions and generalizations*, in Proc. of the Seventeen Nat'l Conf. on Artificial Intelligence, 2000, pp. 90–97.

[18] W. E. Walsh, M. Wellman, and F. Ygge, *Combinatorial auctions for supply chain formation*, in Proc. of ACM Conf on Electronic Commerce (EC'00), October 2000, pp. 260–269.

[19] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason, *Some economics of market-based distributed scheduling*, in Proc. of the Eighteenth Int'l Conf. on Distributed Computing Systems, 1998, pp. 612–621.