Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 02-009

Multi-Agent Negotiation using Combinatorial Auctions with Precedence Constraints

John Collins, Maria Gini, and Bamshad Mobasher

February 18, 2002

Multi-Agent Negotiation using Combinatorial Auctions with Precedence Constraints*

John Collins† and Maria Gini
University of Minnesota

Bamshad Mobasher
DePaul University

Abstract

We present a system for multi-agent contract negotiation, implemented as an auction-based market architecture called MAGNET. A principal feature of MAGNET is support for negotiation of contracts based on temporal and precedence constraints. We propose using an extended combinatorial auction paradigm to support these negotiations. A critical component of the agent negotiation process in a MAGNET system is the ability of a customer to evaluate the bids of competing suppliers. Winner determination in standard combinatorial auctions is known to be difficult, and the problem is further complicated by the addition of temporal constraints and a requirement to complete the winner-determination process within a hard deadline. We introduce two approaches to the extended winner determination problem. One is based on Integer Programming, and the other is a flexible, multi-criterion, anytime bid evaluator based on a simulated annealing framework. We evaluate the performance of both approaches and show how performance data can be used in the agent's deliberation-scheduling process. The results show that coarse problem-size metrics can be effectively used to predict winner-determination processing time.

1 Introduction

Recently, the complexity of logistics involved in manufacturing has been increasing nearly exponentially. Many processes are being outsourced to outside contractors, making supply chains longer and more convoluted. The increased complexity is often compounded by reduced inventories and accelerated production schedules which demand tight integration of processes across multiple self-interested organizations. Firms can cut costs and improve efficiency by moving online. Instead of fulfilling orders from warehouses, companies will look for partners that can provide coordinated components and services in order to meet consumers demand for make-to-order products. Thus, the field is ripe for the introduction of systems that automate logistics planning among multiple entities such as manufacturers, part suppliers and specialized subcontractors.

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

†jcollins@cs.umn.edu

Current e-commerce systems typically rely on either fixed-price catalogs or simple auctions. Companies usually work with pre-qualified suppliers, and buyer-supplier relationships depend on factors such as quality, delivery performance, and flexibility, in addition to cost [14]. In addition, current e-commerce systems do not have any notion of time (except for domain specific systems such as SABRE used in the travel industry). Time plays a fundamental role in supply-chain formation and management, since many products are made up of different parts and require multiple suppliers who have to coordinate their work.

We are interested in studying how a group of heterogeneous, self-interested agents, can make commitments and carry out plans that require multiple tasks and coordination among multiple agents. We have proposed a market architecture [9] and we have implemented prototypes of both the market architecture and the agents. We call this system MAGNET (Multi AGent NEgotiation Testbed). MAGNET provides support for multi-agent negotiations for tasks with temporal and precedence constraints. This is especially important for the coordination of supply-chain management with production scheduling.

One of the more difficult problems an agent faces in dealing with negotiation over collections of tasks is the problem of evaluating bids. Since we use an auction mechanism to carry out the negotiation, the bid-taking agent must solve a combinatorial winner-determination problem. In our case, this requires satisfying temporal feasibility constraints while attempting to minimize cost. We have developed and tested an Integer Programming (IP) formulation of this problem, along with a highly tunable anytime search, based on a Simulated Annealing [25] (SA) framework. In a time-limited negotiation environment, both approaches have value.

This paper is organized as follows: in Section 2 we identify the key requirements for agents and supporting infrastructure that can support time-limited negotiation over coordinated tasks, and we present a novel architecture that satisfies those requirements. Section 3 describes in more detail the decision processes that must be performed by the Customer agent, focusing primarily on the bid-evaluation process. Section 4 describes our hybrid approach to bid evaluation that appears to balance the requirements for anytime performance with the desire for optimal solutions. Section 5 presents the results of a set of experiments that attempt to characterize the winner determination search process. Section 6 relates our work to other published research. Finally, in Section 7, we conclude and discuss further research opportunities.

2 MAGNET Agents and their Environment

MAGNET gives an agent the ability to use market mechanisms (auctions, catalogs, timetables, etc.) to discover and commit resources needed to achieve its goals. We assume that agents are heterogeneous and self-interested, and may be acting on behalf of different individuals or commercial entities who have different goals and different notions of utility.

Agents may fulfill one or both of two roles with respect to the MAGNET architecture, as shown in Figure 1. Customer agents pursue their goals by formulating and presenting Requests for Quotations (RFQs) to Supplier agents through a market infrastructure [8]. The RFQ specifies a task network that includes task descriptions, a precedence network, and possibly other time constraints. Customer agents attempt to satisfy their goals for the

least net cost, where cost factors can include not only bid prices, but also goal completion time and risk factors. More precisely, these agents are attempting to maximize the utility function of some user, as discussed in detail in [3].

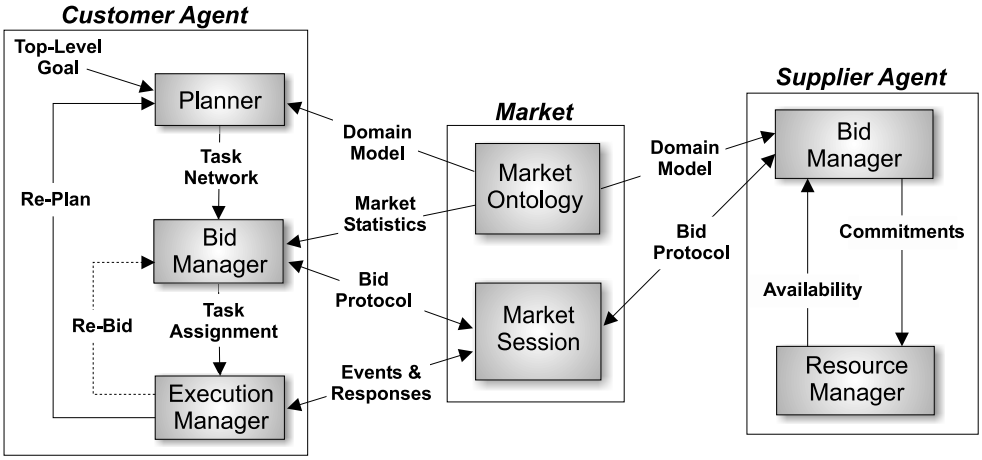

Figure 1: The MAGNET architecture

Supplier agents attempt to maximize the value of the resources under their control by submitting bids in response to those RFQs, specifying what tasks they are able to undertake, when they are available to perform those tasks, and at what price. Suppliers may submit multiple bids to specify different combinations of tasks, with prices and time constraints. Currently, MAGNET supports exclusive OR bids (if a supplier submits multiple bids, only one can be accepted), but other bid semantics are possible [21].

The MAGNET market infrastructure acts as an intermediary and information repository in support of participating agents. Most importantly, it serves an *ontology* of task types along with market statistics. These statistics include expected duration and variability, expected price and variability, resource availability, and lead time. The market also collects statistics on supplier reliability, which customer agents can use to estimate risk during the bid-evaluation process.

As an example, let's imagine we need to construct a garage, which must be completed before the first snowfall. Figure 2 shows a plan to complete the garage. Our plan is complicated by a couple of factors. The special doors must be ordered ahead, and will arrive one week after we order them. They also cannot be left outdoors, and so must be installed immediately when they arrive. Also, there is a housing boom in our area, and carpenters are hard to find on short notice.

In this scenario, we expect that the template for the garage-building plan would come from a library of plan outlines, or it would be created manually. The specific task types with specifications and market statistics would be provided by a construction-services market server. The customer agent's user would fill in specifics of the plan, adding necessary time constraints, adding details, and selecting options for the various tasks. The agent would then formulate an RFQ and present it to the market to request bid responses from suppliers. The level of user interactivity and the balance between user control and agent autonomy are likely to be quite variable among different application domains.

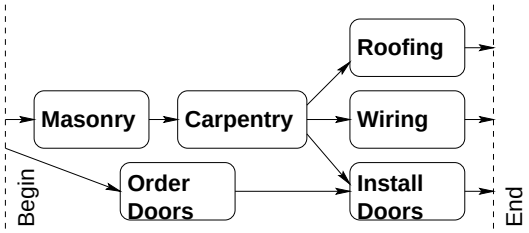

Figure 2: Plan for building a garage

3 Activities of the Customer Agent

We now focus on the structure and responsibilities of a Customer agent in the MAGNET environment. As indicated in Figure 1, the basic operations are planning, bidding, and plan execution. For experimental purposes, we have implemented a simple Planner that generates random task networks with well-defined statistics, and a Bid Manager with a fairly rich set of tools for composing RFQ's and selecting bids. We have also implemented a User Interface that allows manual creation and editing of plans, as well as monitoring and control of customer agent activities during the bid and execution cycles. The Execution Manager has not yet been implemented.

3.1 Task Networks

The Planner's task is to turn the agent's high-level goals into plans, represented as task networks. We have seen an example earlier in Figure 2.

More formally, a task network $\mathcal{P} = (\mathcal{S}, \mathcal{V})$ contains a set $\mathcal{S}$ of task descriptions, and a set $\mathcal{V}$ of precedence constraints. For each task $s \in \mathcal{S}$, there is a possibly empty set $v \in \mathcal{V}$ of precedence constraints $v = \{s' \in \mathcal{S} \mid s' \prec s\}$, the set of tasks s' whose completion must precede the start of task s. Other types of precedence constraints are certainly possible, such as start-start and finish-finish constraints. Notice that the task network itself need not be situated in time. That will happen when we compose the RFQ. Also notice that the operations in the task network do not need to be linearized with respect to time, since operations can be executed in parallel by multiple agents.

The planner in the MAGNET testbed generates tasks by selecting randomly from a library of task types, and then creates random precedence constraints among them. It can also accept pre-defined plans, from a library or from a user interface.

The task network is a central data structure throughout a MAGNET system. The Bid Manager uses it to generate RFQs and to evaluate and record resource commitments and timing data, and the Execution Manager uses it to monitor and repair the ongoing execution of the plan. Part or all of the task network is included in a RFQ. In fact, each of the other components can be characterized by how it uses, decorates, extends, or updates the task network.

3.2 RFQ Generation

The Bid Manager is responsible for ensuring that resources are assigned to each of the tasks of a task network, that the assignments taken together form a feasible schedule, and that the cost of executing the plan is minimized. This cost must also be less than the value of the goal at the time the goal is reached.

When the Bid Manager is invoked, some tasks in the task network may already be assigned. This can occur because the Execution Manager may use the Bid Manager to repair a partially-completed task network in which previously determined assignments have failed, because the agent will perform some of the tasks itself, or because bidding is being carried out in multiple stages. For example, I may decide to wire my new garage myself, and so I might not wish to solicit a contract for that task.

Before constructing the RFQ, the Bid Manager must allocate time to negotiation and plan execution. The timeline in Figure 3 shows an abstract view of the progress of a single negotiation. At the beginning of the process, the Customer agent must allocate deliberation time for its own planning, for supplier bid preparation, and for its own bid evaluation process. Two of these time points, the Bid deadline and the Bid Award deadline, must be commumicated to suppliers as part of the RFQ. The Bid deadline is the latest time a supplier may submit a bid, and the Bid Award deadline is the earliest time a supplier may expire a bid. The interval between these two time points is available to the customer to determine the winners of the auction.

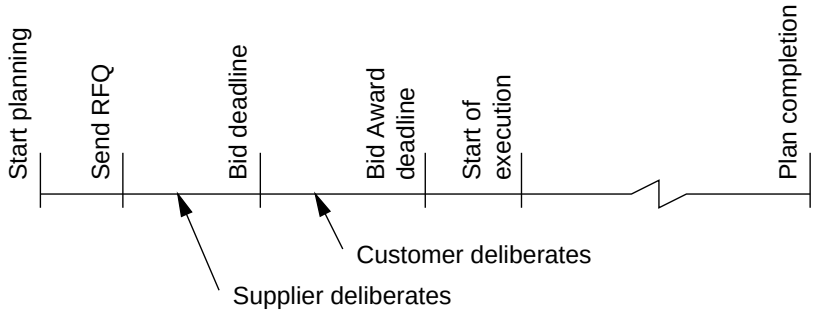

Figure 3: Typical Agent Interaction Timeline

In general, it is expected that bid prices will be lower if suppliers have more time to prepare bids, more time, and more schedule flexibility in the execution phase. Minimizing the delay between the bid deadline and the award deadline will also minimize the supplier's opportunity cost. On the other hand, the Customer's ability to find a good set of bids is dependent on the time allocated to bid evaluation. We are interested in the performance of the winner determination process precisely because we realize that it takes place within a window of time that must be determined ahead of time, and because we expect better overall results if we can maximize the amount of time available to suppliers while minimizing the time required for customer deliberation. These time intervals can be overlapped to some extent, but doing so creates opportunities for strategic manipulation of the Customer by the Suppliers, as discussed in [5].

The Bid Manager attempts to produce an RFQ that will solicit the most advantageous

set of bids possible. It cannot evaluate bids that are not received, nor can it make successful combinations of bids that conflict with one another over precedence constraints. The approach we take is to find a balance between giving maximum flexibility to suppliers, ensuring that the resulting bids will combine feasibly, and ensuring that the job will be completed by the deadline. We do this by setting early-start and late-finish times in the RFQ for each task.

More formally, an RFQ $r = (\mathcal{S}_r, \mathcal{V}_r, \mathcal{W}_r)$ contains a subset $\mathcal{S}_r$ of the tasks in the task network $\mathcal{P}$, with their precedence relations $\mathcal{V}_r$, and time windows $\mathcal{W}_r$ specifying constraints on when each task may be started and completed. As pointed out earlier, there might be elements of the task network $\mathcal{P}$ that are not included in the RFQ. For each task $s \in \mathcal{S}_r$ in the RFQ the Bid Manager must specify:

- a time window $w \in \mathcal{W}_r$, consisting of an earliest start time $t_{es}(s, r)$ and a latest finish time $t_{lf}(s, r)$, and
- a set of precedence relationships $\mathcal{V}_s = \{s' \in \mathcal{S}_r \mid s' \prec s\}$, the set of other tasks $s' \in \mathcal{S}_r$ whose completion must precede the start of s.

There is no requirement that the bidding for a given task network be driven through a single RFQ, and there is no requirement that all precedence relationships be specified in the RFQ. The only requirement is that all specified precedence relationships be among tasks in a single RFQ.

We assume the agent has general knowledge of normal durations of tasks. One of the roles of the MAGNET market infrastructure is to gather and publish statistical information about duration and variability for the different task types, information about resource availability, and about the number of vendors who are likely to bid on each type of task. In order to minimize bid prices while minimizing the overall duration of the task network, the Customer agent schedules tasks ahead of time using expected durations. One straightforward way to do this is to compute early start and late finish times using the Critical Path (CPM) algorithm [15].

The Critical Path algorithm walks the directed graph of tasks and precedence constraints, forward from time t_0 to compute the earliest start $t_{es}(s)$ and finish $t_{ef}(s)$ times for each task $s \in \mathcal{S}$, and then backward from time t_{goal} to compute the latest finish $t_{lf}(s)$ and start $t_{ls}(s)$ times for each task. The minimum duration of the entire task network, defined as $max(t_{ef}(s)) - t_0$, is called the *makespan* of the task network. The difference between t_{goal} and the latest early finish time is called the *total slack* of the task network. If t_{goal} is set equal to $t_0 + makespan$, then the total slack is 0, and all tasks for which $t_{ef}(s) = t_{lf}(s)$ are called *critical* tasks. Paths in the graph through critical tasks are called *critical paths.*

The tradeoff between minimizing task duration and attracting usable bids from suppliers affects how slack should be set. Our method based on CPM can work if adequate approximations of task durations are known, because it will reduce the likelihood of bids being rejected for failure to mesh with the overall schedule, and it will reduce the average bid prices to the extent that it reduces speculative resource commitments on the part of suppliers. It is clearly not optimal, however. Intuitively, this is because the CPM blindly method allocates the additional time to all tasks. Non-critical tasks (tasks not on the critical path) are often

allocated too much time, causing unnecessary overlap in their time windows. Also, no distinction is made between tasks for which large numbers of potential bidders exist, and tasks that are likely to interest only a few bidders.

The optimal setting of RFQ time windows would require detailed knowledge of factors such as likely numbers of bidders and the likely constraint tightness on the resources needed to carry out the tasks. Since the Customer agent cannot know this data precisely, the agent must use approximations. The Market maintains statistics to support this, but there is clearly more work to be done in this area.

For our garage-building scenario, an RFQ might look like Figure 4. It includes all tasks in the task network, along with their precedence constraints. The time windows, however, need not obey the precedence constraints; the only requirement is that the accepted bids obey them. The reason for doing this is to make the RFQ more attractive to suppliers, on the assumption that a supplier is more likely to bid, and likely to submit a lower-cost bid, if it is given greater flexibility in scheduling its resources. This complicates the problem of evaluating bids, as we shall see.

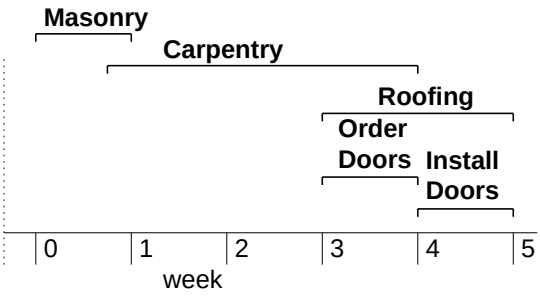

Figure 4: RFQ Example

Many variations on this example are possible. Time windows can be set to avoid any overlap. This guarantees plan feasibility, since every bid will have to fit into a time window and the time windows guarantee the precedence constraints. This in turn will simplify the winner determination process. The disadvantage is a reduction of flexibility for suppliers, since time windows will generally be shorter, or an increase in the time needed to complete the tasks.

An alternative solution is to split the tasks into multiple RFQ's. In our garage example, we might wish to defer requesting bids on the roofing, doors, and door installation until we have bids in hand for the carpentry. If we receive an unexpectedly early bid for the carpentry work, that would allow us to finish the job earlier than if we had built a single schedule, such as the one in Figure 4, with ample slack available to accommodate the expected shortage of carpentry resource availability.

4 Evaluating Bids

Each bid represents an offer to execute some subset of the tasks specified in the RFQ, 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 $\mathcal{S}_b \in \mathcal{S}_r$ 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.

A solution to the bid-evaluation problem is defined as a complete mapping $\mathcal{S} \rightarrow \mathcal{B}$ of tasks to bids in which each task in the corresponding RFQ is mapped to exactly one bid, and that is consistent with the temporal and precedence constraints on the tasks as expressed in the RFQ and the mapped bids.

Figure 5 shows a very small example of the problem the Bid Evaluator must solve. As noted before, 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 large 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.

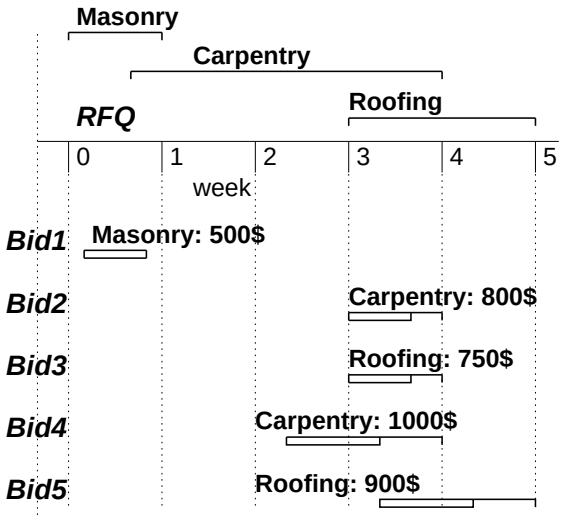

Figure 5: Bid Example

The winner determination problem for combinatorial auctions has been shown to be $\mathcal{NP}$-complete and inapproximable [28]. 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. Because there can be no polynomial-time solution, nor even a polynomial-time bounded approximation, we must accept exponential complexity. We will see in Section 5 that we can determine probability distributions for search time, based on problem size metrics, and we can use

those empirically-determined distributions in our deliberation scheduling process.

A MAGNET Bid Evaluator is a search engine that takes a task network and a set of bids, and attempts to find an optimal or near-optimal complete mapping of tasks to bids, respecting temporal constraints. It must do this within the period of time allocated by the Process Planner, which may have been subdivided by the Bid Scheduler. We have implemented two evaluators: one uses an Integer Programming solver, and the other is a highly modular Simulated Annealing (SA) search engine [6]. We discuss each of them in turn.

4.1 Integer Programming approach

The Integer Programming (IP) solver operates in two phases. The first phase generates basic bid-compatibility constraints, and then walks all paths of length 2 or greater in the precedence network, across all compatible bid combinations, comparing time windows to discover feasibility constraint violations. The second phase composes an IP problem from the resulting constraints and passes the resulting matrix off to an external IP solver[1].

This approach is somewhat different from the "standard" IP formulation for combinatorial auctions (see for example [1]), for two reasons. First, in the MAGNET domain we cannot use the "free disposal assumption" which states that unsold goods can be disposed of freely. In the MAGNET environment, all tasks must be allocated to bids in order to successfully complete the plan. Second, the temporal feasibility constraints dominate the formulation; without preprocessing, relatively small problems can easily generate 10^5 constraints. Preprocessing typically reduces the problem size by 2-3 orders of magnitude.

To describe our IP formulation, we enumerate the set $\mathcal{S}_r$ of tasks in RFQ r as $s_j, j = 1..m$. Each task s_j has a precedence set $\mathcal{V}_j = \{s_{j'} | s_{j'} \prec s_j\}$, the set of tasks $s_{j'} \in \mathcal{S}_r$ that must be completed before s_j starts. At the conclusion of the bidding process, we have a set $\mathcal{B}$ of bids $b_i, i = 1..n$. We use 0/1 integer variables $x_i, i = 1..n$ to denote whether the corresponding bids b_i are included in the result.

Preprocessing starts by discovering uncovered tasks and "singleton" bids. An uncovered task is simply a task s_j that is not included in the task set $\mathcal{S}_i$ of any bid $b_i \in \mathcal{B}$. If an uncovered task exists, then no solution is possible. If there is any task s_j for which only one bid b_i has been received we call bid b_i a "singleton" bid. Any singleton bid b_i must be part of any complete solution, and any 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. If any task becomes uncovered during this process, then no solution exists.

We then discover conflict sets among the remaining bids. For each bid b_i, its conflict set $\mathcal{C}_i$ is defined as the set of bids $b_{i'}$ whose task sets overlap the task set of b_i, i.e.

$$\mathcal{C}_i = \forall b_{i'} \in (\mathcal{B} - b_i) | \mathcal{S}_{i'} \cap \mathcal{S}_i \neq \emptyset$$

[1] For our experiments, we used lp_solve, available from ftp://ftp.ics.ele.tue.nl/pub/lp_solve/

Clearly, only a single bid from any conflict set can be included in any solution.

The final preprocessing step converts precedence relations among tasks to compatibility constraints among bids. For each bid b_i we walk the task precedence network $\mathcal{V}_r$ to discover whether the time windows specified for other bids $b_{i'}$ are compatible with the time windows specified for b_i. Each conflicting time window results in an infeasible bid combination g. Infeasible combinations may be of any length $n = 2 \ldots k$ where k is the maximum depth of the precedence network. They are defined as:

$$\begin{aligned}
\mathcal{G}_2 &= \forall(b_i \in \mathcal{B}, b_{i'} \in (\mathcal{B} - \mathcal{C}_i)) | \{s_j \in \mathcal{S}_i, s_{j'} \in (\mathcal{V}_j \cap \mathcal{S}_{i'}) \\
&\quad | t_{ls}(s_j, b_i) < (t_{es}(s_{j'}, b_{i'}) + d(s_{j'}, b_{i'}))\} \\
\mathcal{G}_3 &= \forall(b_i \in \mathcal{B}, b_{i'} \in (\mathcal{B} - \mathcal{C}_i), b_{i''} \in (\mathcal{B} - \mathcal{C}_i)) \\
&\quad | \{s_j \in \mathcal{S}_i, s_{j'} \in (\mathcal{V}_j \cap \mathcal{S}_{i'}), s_{j''} \in (\mathcal{V}_{j'} \cap \mathcal{S}_{i''}) \\
&\quad | t_{ls}(s_j, b_i) < (t_{es}(s_{j''}, b_{i''}) + d(s_{j''}, b_{i''}) + d(s_{j'}, b_{i'}))\} \\
&\vdots
\end{aligned}$$

The IP formulation is then

Minimize:

$$\sum_{i=1}^{n} c_i x_i$$

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. This is an encoding of the coverage conflict sets discussed above.

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

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

- Feasibility – The final preprocessing step described above produces sets $\mathcal{G}_n$ of bids that cannot be combined feasibly. We must specify that none of these sets can be entirely included in any solution.

$$\forall n \forall g_{n,m} \in \mathcal{G}_n, \sum_{i | b_i \in g_{n,m}} x_i < n$$

More detail on this formulation is available in [4].

```
Procedure simulated_annealing
Inputs:
    S, V: the task network to be assigned
    B: the set of bids
    N_0: initial node, containing mapping of tasks to singleton bids
Output:
    N_best: the node having a mapping M(N_best) = S → B
        of tasks to bids with the best known evaluation
Process:
    Q ← priority queue of maximum length beam_width,
          sorted by node evaluation v(N)
    v(N_0) ← evaluate_node(N_0) /* see narrative in this section */
    N_best ← N_0 /* initialize the result */
    insert(Q, N_0) /* insert the first node into the queue */
    T ← T_0 /* set the initial annealing temperature */
    R ← current_time
    Z ← 0 /* initialize the improvement counter */
    while (¬empty(Q) ∧ R < time_limit) ∧ (Z < patience) do
        N ← select_node(Q, T) /* see Figure 7 */
        B ← select_bid(N, B)
        N' ← expand_node(N, B) /* see Figure 8 */
        v(N') ← evaluate_node(N') /* see narrative in this section */
        insert(Q, N')
        if v(N') < v(N_best) then
            N_best ← N'
            Z ← 0
        else
            Z ← Z + 1
        T ← T(1 − ε) /* the value ε is the annealing rate */
    return N_best
```

Figure 6: Simulated Annealing algorithm for winner determination.

4.2 Simulated Annealing approach

A high level algorithm for our Simulated Annealing winner determination search is depicted in Figure 6. We use a basic simulated-annealing framework [25], with a finite queue (maximum length is *beam_width*) and a number of enhancements to minimize wasted effort. Many complications are omitted, such as the fact that `expand_node` (see Figure 8) may fail to produce a node. This can happen, for example, because the selected bid has already been tried against the selected node, or because of an optional uniqueness test.

The simulated-annealing framework is a queue-based search, characterized by two elements, the annealing temperature T which is periodically reduced by a factor ε, and the

stochastic node-selection procedure shown in Figure 7.

```
Procedure select_node
Inputs:
    Q: the current node queue, sorted so that v(Q_n) ≤ v(Q_{n+1})
        /* v(Q_n) is the evaluation of the n^th node in Q */
    T: the current annealing temperature 0 ≤ T ≤ 1
Output:
    N_next: the selected node
Process:
    r ← random(0, 1) /* a random number uniformly distributed between 0 and 1 */
    R ← v(Q_0) − T ln(1 − r)(v(Q_last) − v(Q_0))
        /* R is called the "target evaluation" */
        /* Q_0 and Q_last are the first and last nodes in the queue, respectively */
    n ← 0
    while v(Q_{n+1}) ≤ R do
        /* find the last node whose evaluation is less than or equal to the target */
        n ← n + 1
    return Q_n
```

Figure 7: The node-selection algorithm.

Stopping conditions include a time limit and "patience factor" as shown in Figure 6, as well as empty queue. The patience factor is simply the number of nodes that have been generated without finding an improved evaluation score. It is typically a function of the problem size, but also may be a function of the number of nodes generated prior to the latest improved score. The queue can become empty if an optional feature is used that keeps track of the bids that have been used to expand the node in the past; when all bids that do not conflict with bids already in the node (where "conflict" can include the conflict sets and feasible bid combinations found for the IP solver) have been tried, the node is removed from the queue. This greatly reduces redundant effort in the search at the cost of keeping track of the remaining compatible bid set per node. Although this is a somewhat unorthodox approach for a stochastic search, it has proven to be very valuable in our situation because of the high cost of node evaluation.

When a stopping condition other than time limit occurs, an optional restart protocol can be used. In most cases, search time is better spent running several shorter searches, rather than a single long search. We shall see an example of this in Section 5.

Bid selection is done by a plug-in component that can be configured in a number of ways. The simplest selector simply makes a random choice among bids that are not part of the current mapping, and that are compatible with the bids currently mapped in the node. Others may focus on improving coverage, feasibility, or cost. Extensive experimentation has failed to show that any more informed method performs consistently better than a simple random approach.

Node expansion (see Figure 8) is done by adding mappings of the selected bid to all the tasks covered by that bid, discarding mappings of any other bids that overlap the new bid. This will fail if the selected bid has already been used to expand the selected node, or because the bid was used to produce the current node. Not shown is the fact that expansion can also fail if an attempt is made to unmap a singleton bid (a bid that provides the only possible mapping for some task). Of course, no bids will ever be unmapped if we filter bid selections with the conflict sets $\mathcal{C}_i$.

Procedure `expand_node`
Inputs:
N: the node to be expanded
B: the bid to be added to $\mathcal{M}(N)$, the bid-task mapping of N
Output:
N': a new node with a mapping that includes B, or null if the mapping fails
Process:
`if` $(B \in \mathcal{B}_{\mathcal{M}(N)}) \bigvee (B \in tried(N))$ `then`
`return null`
/* *B is already in mapping of N, or as been tried previously.* */
$N' \leftarrow$ `copy`(N)
`insert`$(tried(N'), B)$ /* *tried is a set* */
$\mathcal{S}' \leftarrow \mathcal{S}_B \bigcap \mathcal{S}_{\mathcal{M}(N')}$ /* $\mathcal{S}'$ *is the set of tasks in both B and N'.*$\mathcal{M}$ */
$\mathcal{B}' \leftarrow b \in \mathcal{B}_{\mathcal{M}(N')}$ `such that` $\mathcal{S}_b \subset \mathcal{S}'$
/* $\mathcal{B}'$ *is the set of bids in* $\mathcal{M}(N')$ *whose tasks overlap the tasks in B* */
$\forall b \in \mathcal{B}', \forall s \in \mathcal{S}_b, \mathcal{M}(N') \leftarrow \mathcal{M}(N') - \mathtt{m}(s, b)$ /* *remove mappings for* $\mathcal{B}'$ */
$\forall s \in \mathcal{S}_B, \mathcal{M}(N') \leftarrow \mathcal{M}(N') + \mathtt{m}(s, B)$ /* *add the mappings for B* */
`return` N'

Figure 8: The node-expansion algorithm.

Node evaluation produces a value $v(N)$ for node N which is a weighted sum of the following four factors:

- *coverage*: are all the tasks covered? a bid that covers some not yet covered tasks should be preferred over a bid that covers tasks already covered.
- *feasibility*: is the current partial solution feasible? If the Critical Path algorithm finds any negative slack, the current partial solution is not feasible, since it violates some time constraint.
- *cost*: what is the total cost?

5 Experimental Results

Bid evaluation to determine winning bids is a critical part of the Customer agent's behavior, and it is clearly a costly combinatorial problem. We have evaluated many approaches to building a search engine that performs well. It is important to characterize its performance because of the need to allocate time to it in the context of the overall negotiation process, and because failure to produce a result within the allocated time will result in failure of the negotiation. Our goal will be to find the probability distributions of search time based on easily-measured problem metrics. We prefer metrics that can be estimated prior to issuing an RFQ, since that is when deliberation scheduling must be done. This will allow us to schedule the winner-determination deliberation with a known level of confidence in finding a solution.

We report on a set of experiments that give us the necessary probability distribution data for both the IP and SA approaches. There are clearly limits on scalability, and there will be unpredictable situations where good solutions will not be found within any fixed time limit.

5.1 Experimental Setup

The experimental setup consists of a set of randomly-generated task networks, a set of randomly-generated bids for each task network, and a bid evaluator. The bid evaluation process is instrumented to measure both elapsed time and the number of steps performed (equation count for the IP evaluator, node count for the SA evaluator). All experiments were run using a dedicated 1800 MHz Linux machine. Timings are given in wall-clock time. The MAGNET system (including the IP preprocessor and the Simulated Annealing solver) is written in Java, and the IP solver is lp_solve, written in C and available from ftp://ftp.ics.ele.tue.nl/pub/lp_solve/.

5.1.1 Customer Agent: Construct and Issue a Request for Quotes

For these experiments, plans are randomly-generated task networks, with a number of controllable parameters, and the RFQs have a controllable amount of added slack. This slack is intended to give bidders enough flexibility to allow a significant fraction of them to bid. In these experiments, approximately 90% of bidders were able to bid on at least one task. Task Network variables include:

1. Number of tasks.

2. Mix of task types. Task types are characterized by average duration, duration variability, average price, and price variability. Both duration and price are normally distributed, positive values.

3. Branch factor. This is the average number of precedence relationships per task. For example, if a particular task has one predecessor and one successor, then the branch factor for that task is 2.

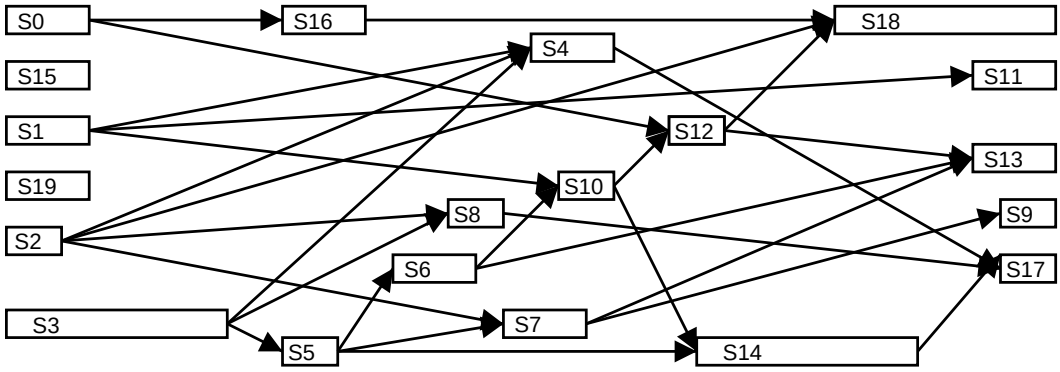

Figure 9: Task network for Problem #1. Box widths indicate relative task durations.

Figure 9 shows the task network for the first of the problems used in the bid-count test series described in Section 5.2.2 below.

Before issuing the RFQ, we compute the makespan for the entire task network, and expected early start times $t_{es}(s)$ and late finish times $t_{lf}(s)$ for each task in the network, as specified in Section 3.2. RFQ variables include:

1. Total slack. This is the ratio $(t_{goal} - t_0)/makespan$ of the time allowed for task network completion to the expected makespan of the task network.

2. Additional slack obtained by reducing task durations below their expected values. For these experiments, we simply multiplied each expected duration by a factor of 0.8 and re-ran the CPM algorithm. The only justification for this number is that it was the greatest reduction we found that did not cause most problems to have no feasible solutions.

5.1.2 Supplier Agent: Generate Bids

For these experiments, we used a test agent that masquerades as an entire community of suppliers. Each time a new RFQ is announced by the Market, the supplier attempts to generate a specified number of bids. For a given RFQ r, each bid b_j is generated by selecting a task s_i at random from $\mathcal{S}_r$, using the task-type parameters to generate a supplier time window $w_i(b_j) = (t_{es}(s_i, b_j), t_{lf}(s_i, b_j), d(s_i, b_j)$ for the task, and testing this time window against the time window $w_i(r)$ for task s_i in the RFQ. If the supplier's time window $w_i(b_j)$ is contained within the RFQ time window $w_i(r)$, then we call this a "valid task specification" and add the task and its time window to the bid. If a valid task specification was generated, then with some probability, each predecessor and successor link from task s_i is followed to choose an additional task to add 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 specifications are not always achieved, some attempts to generate bids will fail.

Task durations are normally distributed around the expected value used by the Customer, and the time windows are randomly set to be smaller than the time windows specified in the

RFQ and larger than the computed durations. Full details on the bid generation process are given in [7].

5.2 IP performance

In this section, and in the following section on SA performance, we attempt to characterize performance along 3 dimensions of problem size: (1) number of tasks in the task network, (2) number of bids submitted, and (3) the sizes of the submitted bids, i.e. the number of tasks per bid. Each problem set consists of 100 problems, with randomly-generated plans and randomly-generated bids as described above. Our problem generator has a large number of parameters; we kept all of them constant except for Task Count, Bid Count, and Bid Size.

5.2.1 Varying task network size

The first set of experiments examines scalability of the search process as the number of tasks varies, with a (nearly) constant ratio of bids to tasks (the ratio varies somewhat due to the random nature of the bid-generation process). Table 1 shows problem characteristics for this set. Each row in the table shows results for 100 problems. In the table, the "Bid Size" column gives the average size of bids (number of tasks per bid). The ratio of Task Count to Bid Size varies a bit because of the way bids are generated: Bid Size is more closely related to the depth of the precedence network than to Task Count. The "Solved" column gives the number of problems solved out of 100 (not all 100 problems were solvable), "Rows" gives the number of constraints in the generated IP problem, "Mean Time" gives the mean search time in milliseconds, including both preprocessor time and time spent in the IP solver itself, and σ gives the standard deviation of the "Total Time" column.

Table 1: Task count experiment for the IP solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Rows	Mean Time (ms)	σ
5	13.5	2.1	94	19.9	16	7
10	29.0	3.3	94	46.3	41	24
15	45.4	4.5	90	84.0	111	90
20	61.0	5.4	85	153	235	172
25	77.7	6.3	85	287	565	633
30	93.4	7.4	80	496	986	859
35	110.6	8.4	73	774	2057	2057

While the data in Table 1 show average performance and give some indication of variability, we are really more interested in knowing the *probability* that a solution can be determined in a given amount of time. For that purpose, we show in Figure 10 the complete runtime distributions for four of the problem sets. For each curve, we show actual observations along

with a lognormal distribution that minimizes the χ^2 metric[2]. Typical χ^2 values range from 0.3 to 3.0 for 9 degrees of freedom, a good match. Note that the first parameter of the lognormal distribution is equal to the median value, which is significantly smaller than the mean for all sets.

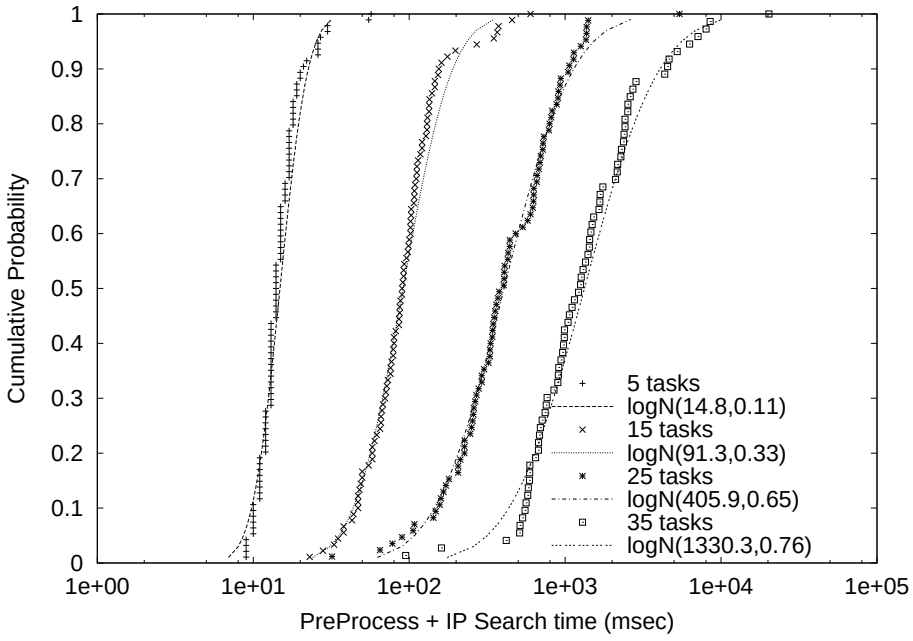

Figure 10: Observed and inferred runtime distributions for the IP solver across a range of task count values, with a nearly constant ratio of bids to tasks.

The value of data such as this is that we can now make decisions about time allocation with a specific degree of confidence. For example, we can say that for a problem the size of our 35-task, 123-bid sample, we have a 95% confidence in finding a solution using the IP solver in less than 5.6 seconds. In order to make use of this sort of data in an agent environment, we need to know how runtime scales along at least 3 dimensions of problem size: task count, bid count, and bid size. We have shown the task count data, but the results are necessarily conflated with variations in bid-size and bid-count. We'll return to this after we have seen the bid count and bid size data.

5.2.2 Varying the number of bids

The next series examines the scalability of the IP method as the number of bids is varied over a 4:1 range, with task count and bid size held constant. Each row in Table 2 represents 100 problems, each with 20 tasks and varying numbers of bids. The same set of 100 task networks is used in each row; only the bid sets are varied. It is clear that the mean time

[2]The χ^2 metric is determined by dividing the inferred density function into a number of equal areas, and counting the number of observations that fall into each of those zones. The χ^2 value is the mean square deviation from a "perfect fit".

scales exponentially with bid count; deviations in mean time and the lower variability at the low end of the range are likely the result of fixed overhead.

Table 2: Bid Count experiment for the IP solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Rows	Mean Time (ms)	σ
20	51.7	7.36	75	92.2	109	61
20	69.6	7.32	95	125	185	95
20	86.5	7.30	94	193	335	228
20	104.4	7.35	96	284	501	452
20	121.0	7.29	98	583	1160	1820
20	139.2	7.26	100	782	1750	1970
20	156.4	7.29	99	1078	3060	3810
20	173.2	7.30	100	1691	4500	4220
20	190.9	7.37	100	2342	8300	15,000
20	207.4	7.32	100	3284	11,800	15,500

In Figure 11, we see the measured and inferred probability distributions for five of the problems from Table 2. Again, we find good correspondence ($0.2 < \chi^2 < 3.0$, 9 dof) between the measured data and the inferred lognormal distributions.

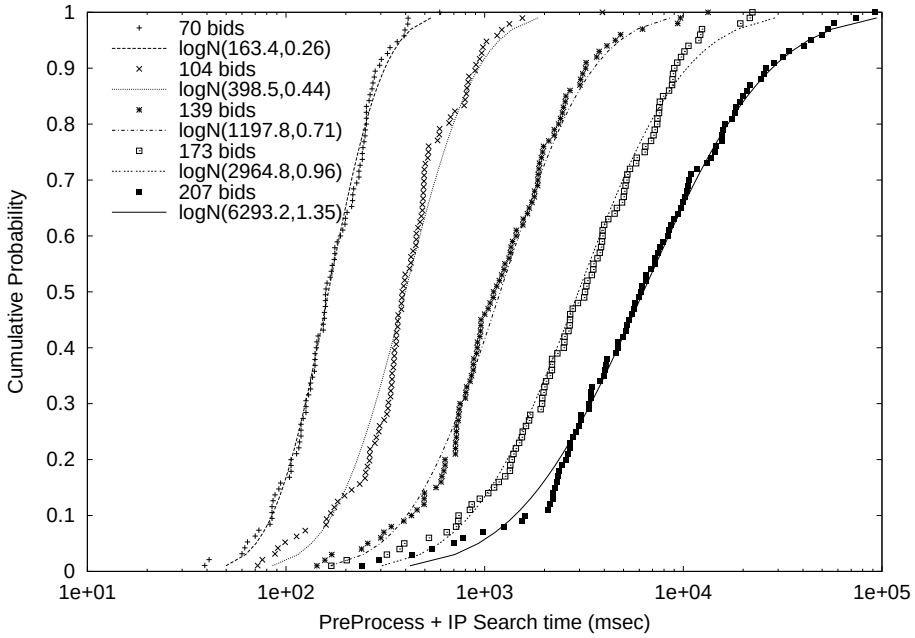

Figure 11: Observed and inferred runtime distributions for IP search for a range of bid count values, with task count = 20, and bid size $\approx$ 7.3.

5.2.3 Varying the sizes of bids

We now turn to the bid-size dimension. Our problem generator uses a somewhat inexact method to "influence" the sizes of bids. As described earlier, the bid-generation process generates "contiguous" bids by choosing a starting point in the task network, and then recursively following predecessor and successor links with some probability. In this series, we have varied that probability from a low of 0.2 to a high of 0.9. Bid size is also influenced by the sizes of time windows, since the "success" in generating a bid for a particular task is a function of both the size of the time window specified in the RFQ, and the simulated "resource availability" recorded for the type of each task. As shown in Table 3, the resulting bid size varies from a low of 1.6 tasks/bid to 7.9 tasks/bid. Note the extreme difficulty and high variability of the 1.6 tasks/bid set. The task networks in these problem sets are the same as the ones used in the bid-count experiment.

Table 3: Bid Size experiment for the IP solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Rows	Mean Time (ms)	σ
20	87.0	1.61	81	8430	87,300	334,000
20	86.6	2.20	92	2440	7550	12,500
20	86.7	3.04	82	1760	5550	13,700
20	86.5	4.22	99	781	1951	2710
20	86.6	5.42	96	374	740	607
20	86.9	6.40	96	259	466	365
20	86.5	7.30	94	193	345	250
20	86.6	7.90	94	164	285	199

Figure 12 shows the runtime observations and inferred distributions for four of the problems from Table 3. It appears that bid size variation affects the variability of solution time as much as it affects the mean time.

5.2.4 Predicting IP runtime

We now have the necessary data to estimate the time that must be allocated to the winner determination process using the IP solver. The process is to choose a desired "probability of success", and then to estimate parameters for a function that reasonably matches the inferred distributions at that level of probability. For example, in the left plot in Figure 13, we show a the 95, 50, and 5 percentile points from the inferred distributions from the bid-count experiment. The overlaid curve is $65.86 \cdot 2^{0.045x}$, which yields a root mean square error $\sqrt{\bar{\epsilon^2}}$ of 376 msec. In the right plot, we show the 95, 50, and 5 percentile points from the bid-size experiment. The overlaid curve in the right plot is $3.75 \times 10^5 \cdot 2^{-1.29x}$, which yields a $\sqrt{\bar{\epsilon^2}}$ of 8.2 sec. It seems likely that actual relationship between bid size and run time is more complex than the simple exponential relationship shown here.

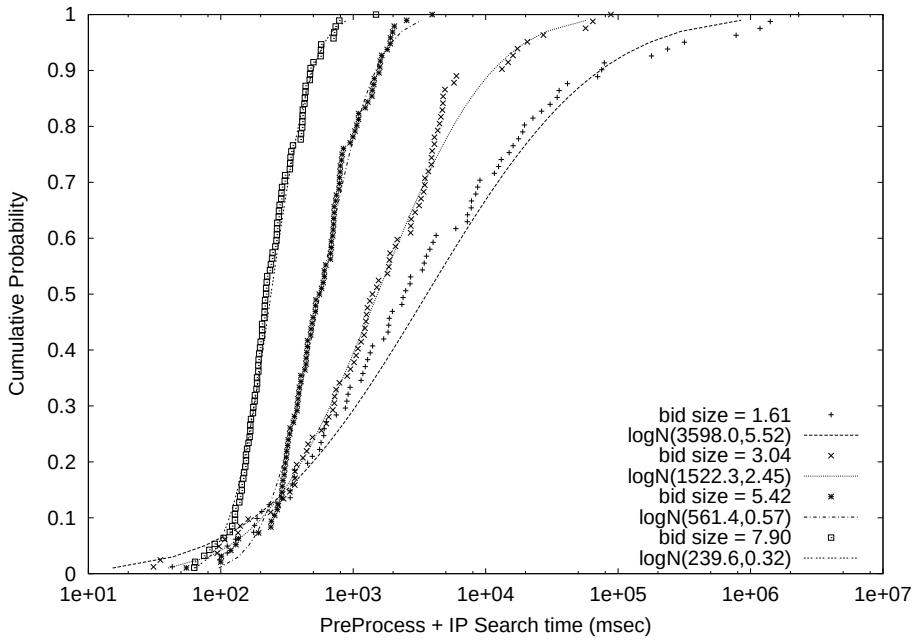

Figure 12: Observed and inferred runtime distributions for IP search for a range of bid size values, with task count = 20, and bid count $\approx$ 87.

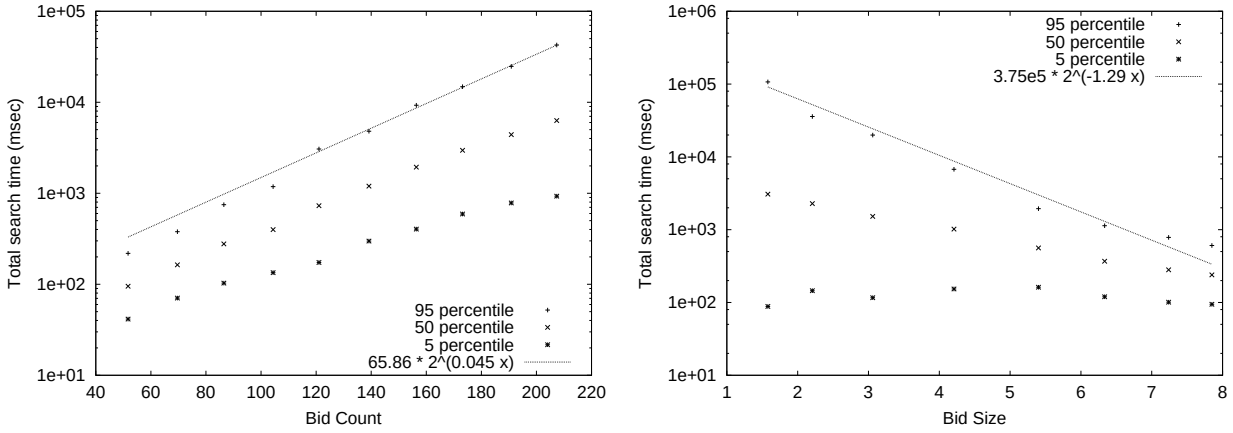

Figure 13: Estimating the time required to achieve 95% success rate using the IP solver for 20-task problems across a range of bid counts (left) and bid sizes (right).

Finally, in Figure 14, we show an attempt to approximate a predictor for the search time required for the task-count experiment as detailed in Table 1. In addition to varying the task count, this experiment varies the bid count and the ratio of bid size to task count. We have taken the coefficients derived for bid count and bid size from above, and derived a coefficient for task count that minimizes $\bar{\epsilon^2}$. The derived formula is

$$runtime = 13.30 \cdot 2^{(-25.8\frac{bs}{tc}+0.045bc+0.00384tc)}$$

where tc is the task count, bc is the bid count, and bs is the bid size. The error $\sqrt{\bar{\epsilon^2}}$ for these coefficients is 71.5 msec. The search times on the 5-task set are so short they are very

likely to have a large fixed overhead component (the lp_solve process was re-started for each search). They almost certainly also suffer from the limited resolution of the system clock.

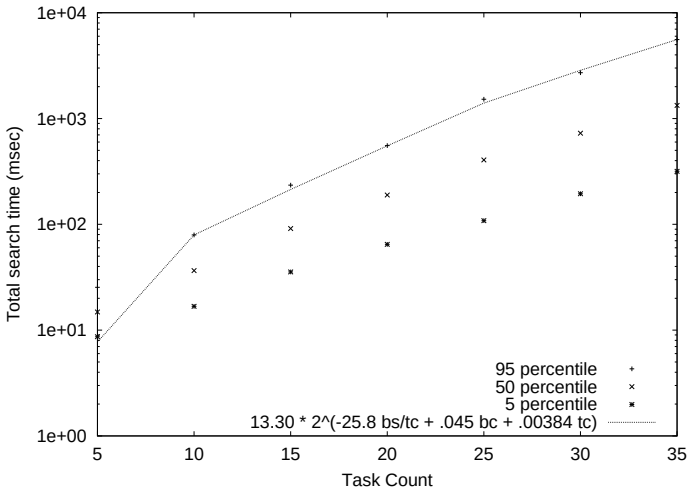

Figure 14: Estimating IP search time for the task count experiment.

5.3 Characterizing the SA evaluator

Since the Simulated Annealing solver is a stochastic algorithm, we are interested in two types of performance data. The first is the same type of data that we saw for the IP solver in the last section: performance across a range of task counts, bid counts, and bid sizes. The second is a more detailed look at the variability observed on individual problems.

5.3.1 Varying task network size

In Table 4 we report data on SA performance as task count varies. This is the same set of problems that was reported in Table 1 on IP performance, so bid count and bid size data can be read from that table. As before, we generated 100 problems in each set, and ran the IP solver to find the optimal solution (or to determine that the problem could not be solved). For each "solvable" problem we then ran the SA solver 20 times with different random number sequences, and captured time and node count data when the solver first found solutions with evaluations within 5% of the optimal value, within 1% of optimal, and equal to the optimal value found by the IP solver. In the remainder of this section, we'll call these "5% solutions," "1% solutions," and "optimal solutions," respectively. We then averaged the values for successful attempts on each problem, and we show here the statistics for those average values across the various problem sets.

In Table 4, the "Failure Rate" column gives the proportion of attempts in which the SA solver failed to find optimal solutions to solvable problems (problems solved by the IP solver). The "Node Time" column reports the mean time per search node generated. These times grow as problem size increases because of the cost of maintaining the sorted search queue. Then we give Mean, Median, and Standard Deviation data for the time in milliseconds

required to find a 5% solution and an optimal solution. We give both Mean and Median here because they are substantially different, due to the skewed distribution.

Table 4: Task count experiment for the SA solver

Task Count	Failure Rate (%)	Node Time (ms)	5% Solution Mean (ms)	5% Solution Median (ms)	5% Solution σ	Optimal Solution Mean (ms)	Optimal Solution Median (ms)	Optimal Solution σ
5	0	0.23	5	4	4	7	5	5
10	0	0.31	92	75	82	250	116	915
15	2	0.36	392	282	442	2270	652	7610
20	4	0.42	2000	857	4400	12,600	2550	29,500
25	11	0.48	9420	2420	20,300	26,500	6610	41,300
30	19	0.54	19,600	4270	45,600	50,800	11,300	93,800
35	23	0.60	35,700	8220	72,500	97,900	27,900	126,000

Figure 15 shows graphically the observed distributions for 4 of the problem sets shown in Table 4. Each individual data point represents the average time required to find an optimal solution using the SA solver. Only the successful trials were included, so there is likely some truncation at the upper end of the range on the larger problems. We also show the closest lognormal curves, but clearly the data is not as well-behaved as the IP data, and doesn't fit a lognormal distribution for the larger problems.

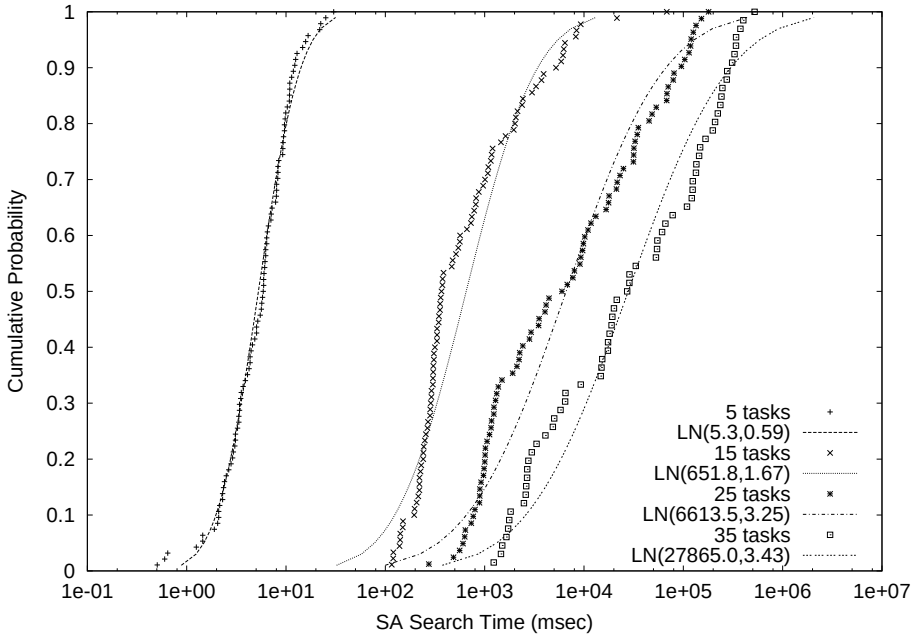

Figure 15: Observed and lognormal runtime distributions for SA search to find optimal solutions across a range of task count values.

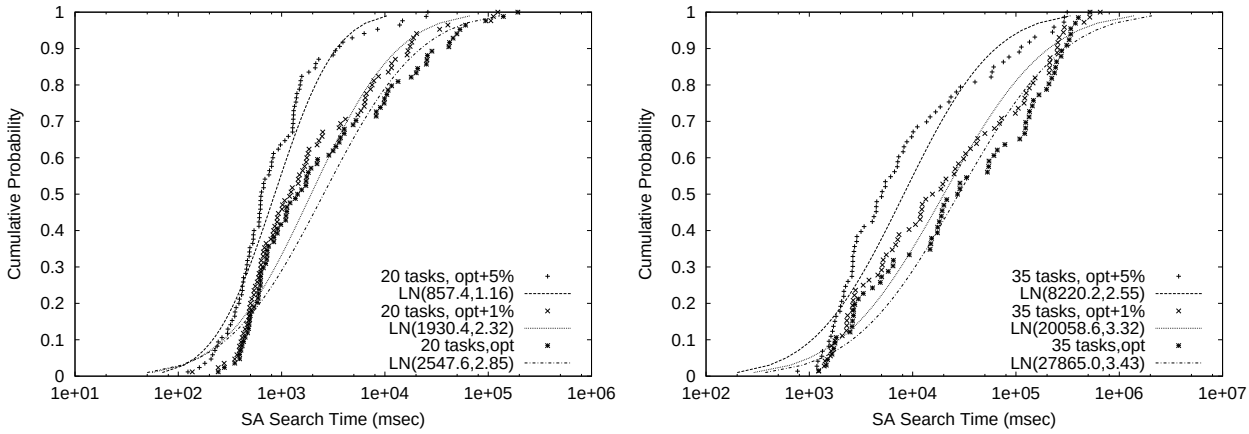

Figure 16: Observed distributions for finding 5%, 1%, and optimal solutions using Simulated Annealing, on problems having 20 tasks and 35 tasks.

In a time-limited negotiation situation, we might be willing to settle for a solution that is less than optimal. An obvious question is how much time we gain by doing so. Figure 16 shows the detailed aggregate behavior for two of the problem sets in Table 4. Again, each point represents the mean time required to find solutions to the indicated level of accuracy, for trials in which solutions were found. There is some degree of truncation in the data for finding optimal solutions in the 35-task case: the overall failure rate was 22.5%, and there were 7 solvable problems for which the SA solver never found optimal solutions. The failure rates for the 5% and 1% solutions were 2.6% and 13.3% respectively, and there was one problem for which no 1% solution was found. For the 20-task problems, there were no failures to find the 5% solution, a 2.1% failure rate for finding 1% solutions, and a 3.8% failure rate for finding optimal solutions.

5.3.2 Varying the number of bids

Table 5 shows SA performance results on a subset of the problems for which we reported IP results in Table 2. The long runtimes we encountered precluded us from running the full 20-search/problem SA experiment on all 10 of the problem sets in the bid count experiment. In addition, the 156-bid set is significantly truncated, and we assume that the level of truncation in the larger sets would have been even greater.

In Figure 17 we see the observed runtime distributions for these four sets. We also show the "closest" lognormal curves, as determined by computing maximum likelihood estimators, for comparison purposes. Whereas the lognormal curves were a good match for the characteristics of the IP runtime distributions, they are clearly not a good match here. These data are not well-behaved, and for the larger problems the high end is truncated. This happens because we have placed limits on the ability of SA to search, including limiting the queue size and the number of restarts, and by running the annealing temperature down to a level where there is little randomness (typically the end temperature is between 10^{-2} and 10^{-3}) at the end of each restart attempt.

Table 5: Bid count experiment for the SA solver

Bid Count	Failure Rate (%)	Node Time (ms)	5% Solution Mean (ms)	5% Solution Median (ms)	5% Solution σ	Optimal Solution Mean (ms)	Optimal Solution Median (ms)	Optimal Solution σ
51.7	0	0.41	508	352	572	3660	787	11,600
86.5	7	0.43	4490	1130	11,200	24,500	3840	48,200
121.0	16	0.46	13,000	2910	39,000	52,600	12,000	78,600
156.4	31	0.46	37,400	7850	72,900	111,000	36,300	117,000

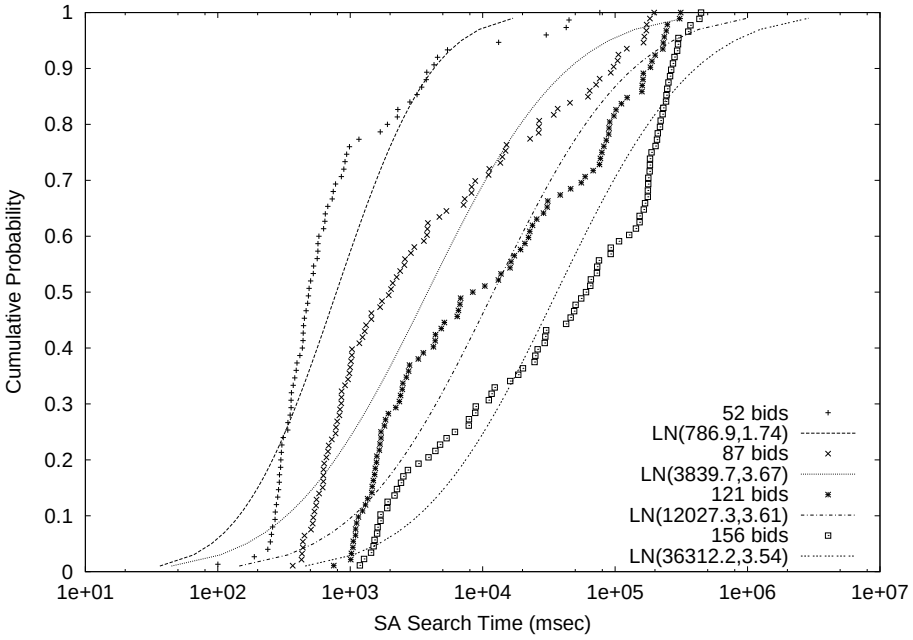

Figure 17: Observed and lognormal runtime distributions for SA search to find optimal solutions across a range of bid count values.

5.3.3 Varying the sizes of bids

In Table 6 we see a summary of results from running the SA solver on 8 sets of 100 problems in which the bid size is varied between sets. These are the same problem sets used for the IP bid-size experiments as summarized in Table 3. For the 1.61 tasks/bid set, the mean length of the optimal solutions (number of bids in the solution) was 14.2. For solutions of length 10 or less, the SA solver found optimal solutions 95% of the time. For solutions longer than 14 bids, SA found no optimal solutions, although there were no problems for which at least a 5% solution was not found on at least 3 of the 20 attempts.

Figure 18 shows observed runtime distributions of four of the problem sets from Table 6. We also show "inferred" lognormal distribution curves, although they are clearly not a good match. In particular, the failure rate for the 3.04 tasks/bid case was over 50%. Since

Table 6: Bid size experiment for the SA solver

Bid Size	Failure Rate (%)	Node Time (ms)	5% Solution Mean (ms)	5% Solution Median (ms)	5% Solution σ	Optimal Solution Mean (ms)	Optimal Solution Median (ms)	Optimal Solution σ
1.61	93	0.43	113,000	75,200	68,900	126,000	79,600	103,000
2.20	74	0.42	61,800	29,500	56,400	127,000	90,600	69,600
3.04	57	0.43	34,200	12,600	46,100	116,000	73,300	71,900
4.22	28	0.45	18,000	5740	28,100	70,900	22,800	77,500
5.42	12	0.44	9710	2320	24,200	42,900	10,600	64,700
6.40	9	0.43	4270	1430	10,100	32,300	6620	54,500
7.30	7	0.43	4490	1130	11,200	24,500	3840	48,200
7.90	6	0.43	3070	1080	8490	19,400	3250	40,800

we arbitrarily stopped the SA solver after 100 retries, it is reasonable to expect that the solution times for the unsolved problems would all have been larger than the observed times for solved problems. Therefore the 3.04 tasks/bid curve should probably be plotted only up to the 43% probability coordinate. This would put the median time value much higher. Other than the poorly-defined shapes of these distributions, the striking factor is that they have a very different character from the distributions for the IP solver on the same problems (see Figure 12).

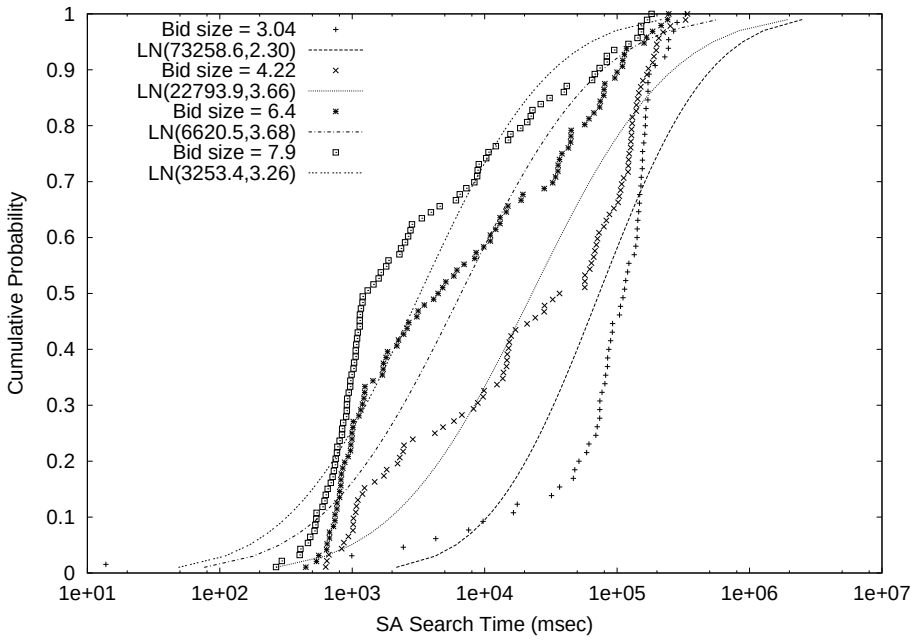

Figure 18: Observed and lognormal runtime distributions for SA search to find optimal solutions across a range of bid sizes. All problems have 20 tasks, 100 bids.

5.3.4 Comparing IP and SA performance

The previous tables and figures describe aggregate performance of the two solvers, and they allow us to confirm that, for an arbitrary problem in a time-limited situation, we have a higher probability of finding an optimal solution using the IP solver than using the SA solver. This is not a surprise. However, this does not tell us that the IP solver is *always* faster. In Figures 19 and 20 we show the relative performance of the IP and SA solvers on each of the individual problems in the 20-task and 35-task sets from the task-count experiment series. In each figure, the left plot compares the IP time with the time required for SA to find a solution within 5% of optimal, and the right plot compares IP time with time for SA to find optimal solutions. For problems below the $x = y$ lines, SA was faster; above the line, IP was faster.

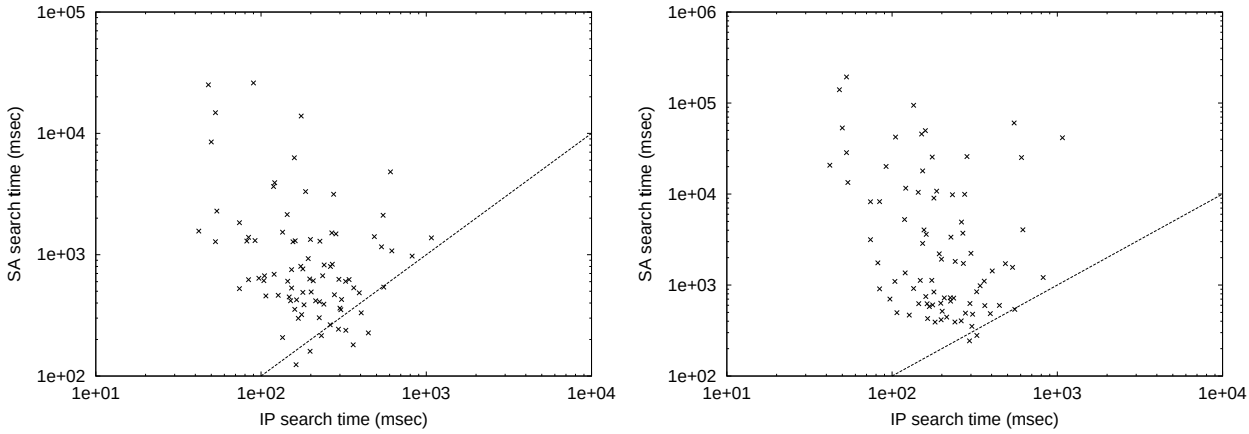

Figure 19: Problem-by-problem comparison between IP solution time and time required by SA to find optimal+5% solutions (left) and optimal solutions (right), 20-task problem set.

There are two interesting features in these plots:

1. The IP search time is not correlated with the SA search time.

2. For larger problems and for lower demands on optimality, SA outperforms IP in many cases.

Figure 21 shows similar data for the 2.2 bids/task problem set from the bid-size experiment. In this case the difference between the two is even more striking. We can easily see the large variability of the IP solution time, and some of the SA solutions were more than an order of magnitude faster than the IP solution for the same problems.

We have not attempted to develop runtime predictors for the SA solver as we did in Section 5.2.4 for the IP solver. This is partly because we have not found probability distributions that are a good match for the runtime characteristics of the SA solver, but more importantly it is because there are many tuning parameters for the SA solver, and these data represent just one possible combination of settings of those parameters. If the goal is to use the SA solver to "hedge" the runtime of the IP solver, one would likely use very different parameter settings, and accept a higher failure rate. For a final comparison, we

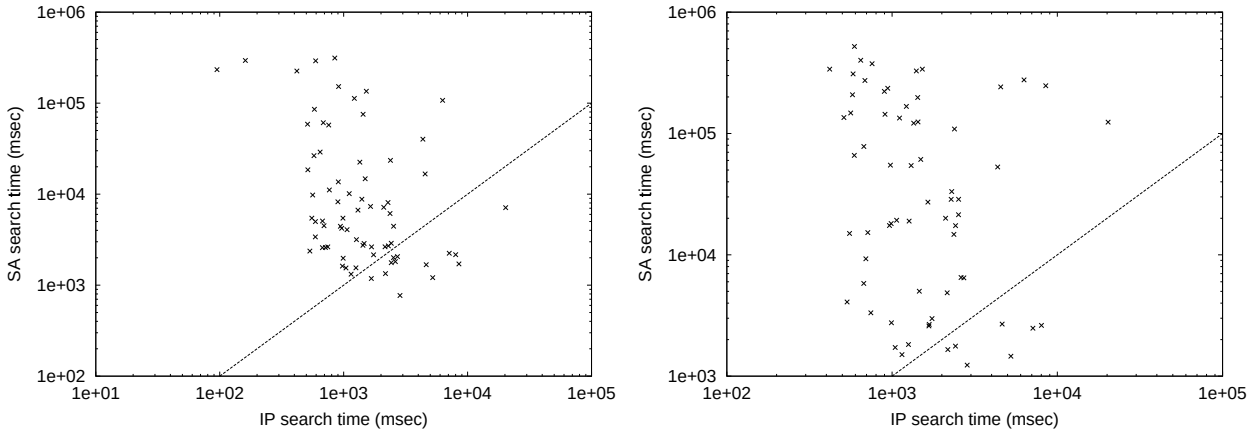

Figure 20: Problem-by-problem comparison between IP solution time and time required by SA to find optimal+5% solutions (left) and optimal solutions (right), 35-task problem set.

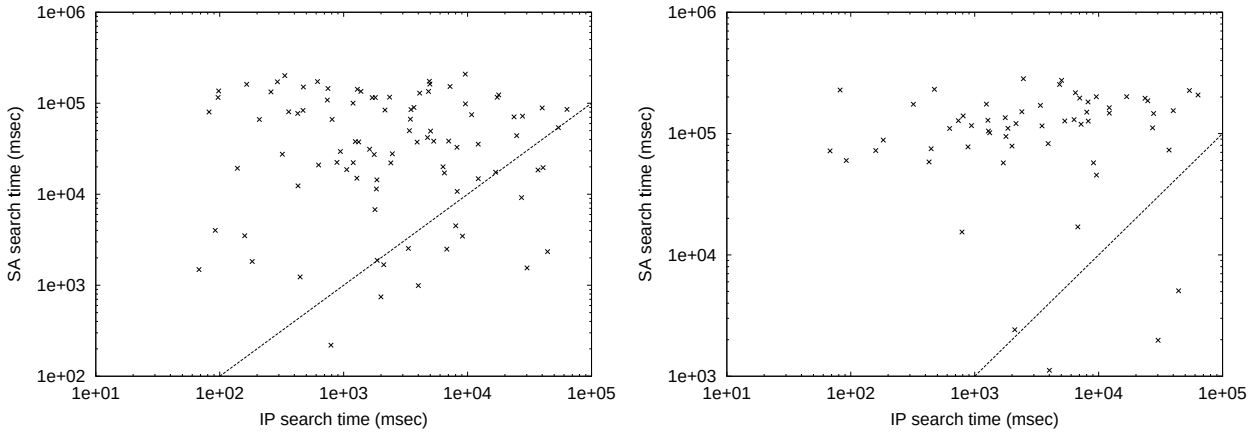

Figure 21: Problem-by-problem comparison between IP solution time and time for SA to find optimal+5% solutions (left) and optimal solutions (right), 20 tasks, 2.2 tasks/bid.

show in Figure 22 the result of changing two of the basic SA parameter settings. The left plot is the same as the left plot in Figure 19, in which the "patience factor" is set to 80, and the "beam-width" is set to 200. As discussed earlier, the patience factor controls the number of iterations without finding an improved node that will trigger termination of a restart cycle, and the beam width is a limit on the size of the search queue. Both factors are scaled by problem size; these are the settings used in all the SA runs described so far. The right plot in Figure 22 shows the same problem set, the same IP data, but we have set the patience factor to 40 and the beam width to 50. The number of problems for which SA finds its "optimal+5%" solution faster than IP finds its solution is nearly doubled. However, the failure rate for the SA solver under these conditions jumps from 3.8% in the original set to nearly 15% in the "faster" set.

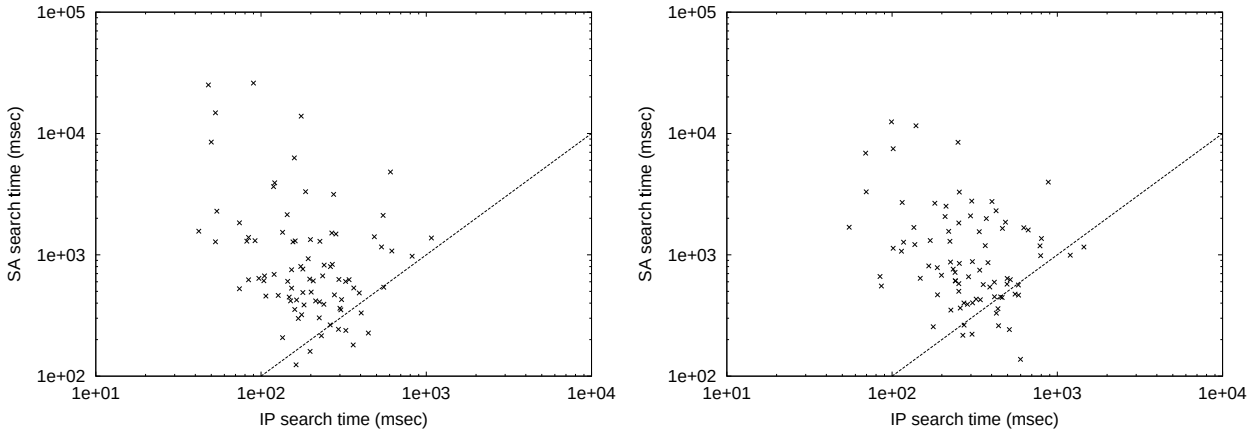

Figure 22: Problem-by-problem comparison between IP time and time for SA to find optimal+5% solutions, 20-tasks, SA parameters: pf=80, bw=200 (left); pf=40, bw=50 (right).

6 Related Work

Auctions are becoming the predominant mechanism for agent-mediated electronic commerce [12]. AuctionBot [34] and eMEDIATOR [28] are among the most well known examples of multi-agent auction systems. They use economics principles to model the interactions of multiple agents [32]. Simple auctions are not always the most appropriate mechanism for the business-to-business transactions we are interested in, where issues such as scheduling, reputation, and maintaining long term business relations are often more important than cost.

Existing architectures for multi-agent virtual markets typically rely on the agents themselves to manage the details of the interaction between them, rather than providing explicit facilities and infrastructure for managing multiple negotiation protocols. In our work, agents interact with each other through a market. The independent market infrastructure provides a common vocabulary, collects statistical information that helps agents estimate costs, schedules, and risks, and acts as a trusted intermediary during the negotiation process. We believe there are many advantages to be gained by having a market-based intermediary for agent negotiations, as explained earlier in the paper.

Rosenschein and Zlotkin [26] showed how the behavior of the agents can be influenced by the set of rules that the system designers choose for the agents' environment. In their study the agents are homogeneous and there are no side payments. In other words, the goal is to share the work, not to pay for work. They also assume that each agent has sufficient resources to handle all the tasks, while we assume the contrary.

Sandholm's agents [31, 30] redistribute work among themselves by a contracting mechanism. Sandholm considers agreements involving explicit payments, but he also assumes that the agents are homogeneous – they have equivalent capabilities, and any agent can handle any task. Our agents are heterogeneous, and decide what tasks to handle.

The determination of winners of combinatorial auctions [20] is hard. Dynamic programming [27] works well for small sets of bids, but does not scale and imposes significant restrictions on the bids. Sandholm [28] relaxes some of the restrictions and presents an algorithm for optimal selection of combinatorial bids, but his bids include only cost. A set

of optimal and approximate methods, along with a test set for algorithm evaluation, was published by Fujishjima et al [10]. Hoos and Boutilier[16] describe a stochastic local search approach to solving combinatorial auctions, and characterize its performance with a focus on time-limited situations. A key element of their approach involves ranking bids according to expected revenue; it's very hard to see how this could be adapted to the MAGNET domain with temporal and precedence constraints, and without free disposal. Andersson et al [1] describe an Integer Programming approach to the winner determination problem in combinatorial auctions. Nisan [22] extended this with an analysis of bidding languages for combinatorial auctions. More recently, Sandholm [29] has described an improved winner-determination algorithm called CABOB that uses a combination of linear programming and branch-and-bound techniques. It is not clear how this technique could be extended to deal with the temporal constraints in the MAGNET problem, although the bid-graph structure may be of value.

Because of a desire to reduce the variability of the Integer Programming approach, and the need for agents to make decisions according to a strict timeline, we have evaluated a Simulated Annealing framework as an alternative method. Since the introduction of iterative sampling [18], a strategy that randomly explores different paths in a search tree, there have been numerous attempts to improve search performance by using randomization. Randomization has been shown to be useful in reducing the unpredictability in the running time of complete search algorithms [11], although in our application that variability remains quite high.

A variety of methods that combine randomization with heuristics have been proposed, such as Least Discrepancy Search [13], heuristic-biased stochastic sampling [2], and stochastic procedures for generating feasible schedules [23], just to name a few. The algorithm we present is based on simulated annealing, and as such combines the advantages of heuristically guided search with random search.

Our ultimate goal is to automate the scheduling/execution cycle of a single autonomous agent that needs the services of other agents to accomplish its task. Pollack's DIPART system [24] and the Multiagent Planning architecture (MPA) [33] assume multiple agents that operate independently but all work toward the achievement of a global goal. Our agents are trying to achieve their own goals and to maximize their profit; there is no global goal.

7 Conclusions and Future Work

Auction mechanisms are an effective approach to negotiation among groups of self-interested economic agents. We are particularly interested in situations where agents need to negotiate over multiple factors, including not only price, but task combinations and temporal factors as well. The MAGNET system provides an environment for evaluating such agents.

Winner determination for combinatorial auctions is an important, difficult, combinatorial problem. Adding temporal constraints adds another layer of difficulty. Agents that operate in the MAGNET environment must perform such evaluations within constrained time intervals in order to meet their commitments to other agents. We have described both an Integer Programming approach and a Simulated Annealing approach to this problem. The Integer Programming approach gives substantially better average performance, but its variability

requires looking for further alternatives for the bid evaluation process in a MAGNET agent. Ultimately, the value of the stochastic approach in this application is not that it finds results faster or more reliably than Integer Programming; it does not. Rather, its value is in the fact that its performance characteristics are different; problems that are hard for the IP solver are sometimes relatively easy for the SA solver. This means that an agent that uses both techniques, possibly in parallel, has a better chance of finding a usable solution to the winner determination problem in a limited amount of time. In addition, it is possible to optimize over non-linear preferences using the SA solver. An example would be to prefer a distribution of schedule slack based on estimates of task duration variability and supplier reliability.

We have shown how we can characterize the time requirements for the winner-determination search based on easily-measurable or estimable characteristics of the overall problem: task count, bid count, and bid size. Other measurable characteristics may also turn out to have predictable impact on search effort. Examples might be the density of precedence links in the task network, or the proportion of precedence relations in the task network for which infeasibility is allowed by the RFQ time window specifications. It may also be interesting to characterize the performance variability of the SA solver on individual problems in more detail.

We have implemented the core elements of the MAGNET market infrastructure and of the software agents that operate in that environment. The implementation is in Java, using industry-standard open protocols for communication between the market and the agents. In the future, we intend to benchmark the performance of MAGNET agents against human decision-makers, and to provide an environment in which agents of different designs may be tested against each other. We believe that the most productive, realistic approach to agent design in a commercial contracting environment will be a mixed-initiative approach, in which the agent's responsibility is to act as a decision-support tool and intermediary for a human decision maker. We are also exploring how modifying the parameters of the market and session protocols can effect the performance of the system. It is our intention to make the MAGNET software available to other researchers, and to open our testbed to participation over the Internet.

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. IEEE Computer Society Press, July 2000.

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

[3] John Collins, Corey Bilot, Maria Gini, and Bamshad Mobasher. Mixed-initiative decision support in agent-based automated contracting. In *Proc. of the Fourth Int'l Conf. on Autonomous Agents*, pages 247–254, June 2000.

[4] 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.

[5] John Collins, Scott Jamison, Maria Gini, and Bamshad Mobasher. Temporal strategies in a multi-agent contracting protocol. In *AAAI-97 Workshop on AI in Electronic Commerce*, July 1997.

[6] John Collins, Rashmi Sundareswara, Maria Gini, and Bamshad Mobasher. Bid selection strategies for multi-agent contracting in the presence of scheduling constraints. In A. Moukas, C. Sierra, and F. Ygge, editors, *Agent Mediated Electronic Commerce II*, volume LNAI1788. Springer-Verlag, 2000.

[7] John Collins, Rashmi Sundareswara, Maksim Tsvetovat, Maria Gini, and Bamshad Mobasher. Search strategies for bid selection in multi-agent contracting. In *IJCAI Workshop on Agent-Mediated Electronic Commerce*, Stockholm, Sweden, July 1999.

[8] John Collins, Maxim Tsvetovat, Bamshad Mobasher, and Maria Gini. Magnet: A multi-agent contracting system for plan execution. In *Proc. of SIGMAN*, pages 63–68. AAAI Press, August 1998.

[9] 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.

[10] 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.

[11] Carla Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proc. of the Fifteen Nat'l Conf. on Artificial Intelligence*, pages 431–437, 1998.

[12] Robert H. Guttman, Alexandros G. Moukas, and Pattie Maes. Agent-mediated electronic commerce: a survey. *Knowledge Engineering Review*, 13(2):143–152, June 1998.

[13] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proc. of the 14th Joint Conf. on Artificial Intelligence*, pages 607–613, 1995.

[14] S. Helper. How much has really changed between us manufacturers and their suppliers. *Sloan Management Review*, 32(4):15–28, 1991.

[15] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research.* McGraw-Hill, 1990.

[16] 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.

[17] Holger H. Hoos and Thomas Stützle. Evaluating las vegas algorithms – pitfalls and remedies. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 238–245. Morgan Kaufmann Publishers, 1998.

[18] Pat Langley. Systematic and nonsystematic search strategies. In *Proc. Int'l Conf. on AI Planning Systems*, pages 145–152, College Park, Md, 1992.

[19] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, second edition, 1991.

[20] R. McAfee and P. J. McMillan. Auctions and bidding. *Journal of Economic Literature*, 25:699–738, 1987.

[21] Noam Nisan. Bidding and allocation in combinatorial auctions. Technical report, Institute of Computer Science, Hebrew University, 2000.

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

[23] Angelo Oddi and Stephen F. Smith. Stochastic procedures for generating feasible schedules. In *Proc. of the Fourteenth Nat'l Conf. on Artificial Intelligence*, pages 308–314, 1997.

[24] Martha E. Pollack. Planning in dynamic environments: The DIPART system. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press, 1996.

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

[26] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter*. MIT Press, Cambridge, MA, 1994.

[27] Michael H. Rothkopf, Alexander Pekeč, and Ronald M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

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

[29] 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.

[30] Tuomas W. Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents*. PhD thesis, Department of Computer Science, University of Massachusetts at Amherst, 1996.

[31] Tuomas W. Sandholm and Victor Lesser. On automated contracting in multi-enterprise manufacturing. In *Distributed Enterprise: Advanced Systems and Tools*, pages 33–42, Edinburgh, Scotland, 1995. was indicated as Sand95 in earlier bib file.

[32] Michael P. Wellman and Peter R. Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24:115–125, 1998.

[33] David E. Wilkins and Karen L. Myers. A multiagent planning architecture. In *Proc. Int'l Conf. on AI Planning Systems*, pages 154–162, 1998.

[34] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Second Int'l Conf. on Autonomous Agents*, pages 301–308, May 1998.