MAGNET: A Multi-Agent System using Auctions with Temporal and Precedence Constraints[⋆]

John Collins and Maria Gini

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

Abstract. We consider the problem of rational, self-interested, economic agents who must negotiate with each other in order to carry out their plans. Customer agents express their plans in the form of task networks with temporal and precedence constraints. The market runs a combinatorial reverse auction, in which supplier agents submit bids specifying prices for combinations of tasks, along with time windows and duration data that the customer may use to compose a work schedule. The presence of temporal and precedence constraints among the items at auction requires extensions to the standard winner-determination procedures for combinatorial auctions, and the use of the enhanced winner-determination procedure within the context of a real-time negotiation requires that we predict its runtime when planning the negotiation process.

We address two specific issues related to this problem. The first is the need for a market infrastructure to support decision processes. We propose a set of requirements for a market that can support this type of negotiation, and describe an architecture that can meet these requirements. We also describe the high-level design of an agent that can act as a customer in this environment, and discuss the decision behaviors such an agent must implement to maximize its utility. The second issue we consider is the determination of auction winners. We explore and characterize a winner determination method, which is an extension of the bidtree-based Iterative-Deepening A* (IDA*) formulation proposed by Sandholm.

1 Introduction

We believe that much of the commercial potential of the Internet will remain unrealized until a new generation of autonomous systems is developed and deployed. A major problem is that the global connectivity and rapid communication capabilities of the Internet can present an organization with vast numbers

⋆ Work supported in part by the National Science Foundation under grants IIS-0084202 and IIS-0414466 of new opportunities, to the point that users are overwhelmed, and conventional automation is insufficient.

Much has been done to enable simple buying and selling over the Internet, and systems exist to help customers and suppliers find each other, such as search engines, vertical industry portals, personalization systems, and recommender engines. However, many business operations are much more complex than the simple buying and selling of individual items. We are interested in situations that require coordinated combinations of goods and services, where there is often some sort of constraint-satisfaction or combinatorial optimization problem that needs to be solved in order to assemble a "deal." Commonly, these extra complications are related to constraints among task and services, and to time limitations. The combinatorics of such situations are not a major problem when an organization is working with small numbers of partners, but can easily become nearly insurmountable when "opened up" to the public Internet.

We envision a new generation of systems that will help organizations and individuals find and exploit opportunities that are otherwise inaccessible or too complex to seriously evaluate. These systems will help potential partners find each other (matchmaking), negotiate mutually beneficial deals (negotiation, evaluation, commitment), and help them monitor the progress of distributed activities (monitoring, dispute resolution). They will operate with variable levels of autonomy, allowing users to delegate or reserve authority as needed, and they will provide users with a market presence and power that is far beyond what is currently achievable with today's telephone, fax, web, and email-based methods. We believe that an important negotiation paradigm among these systems will be market-based combinatorial auctions, with added precedence and temporal constraints.

The Multi-AGent NEgotiation Testbed (MAGNET) project represents a first step in bringing this vision to reality. MAGNET provides a unique capability that allows self-interested agents to negotiate over complex coordinated tasks, with precedence and time constraints, in an auction-based market environment. This paper introduces many of the problems a customer agent must solve in the MAGNET environment, explores in detail the problem of solving the extended combinatorial-auction winner determination problem.

This paper is organized as follows. Section 2 works through a complete interaction scenario with an example problem, describing each of the decision processes a customer agent must implement in order to maximize the expected utility of its principal. Section 3 focuses on one specific decision problem, that of deciding the winners in a MAGNET auction. We describe an optimal tree search formulation algorithm for this problem. Section 4 briefly describes the results of experiments that characterize the performance of our search algorithm. This is important because it is a difficult combinatorial problem, and the negotiation process requires that time be allocated to it before the details of the problem can be known. Section 5 places this work in context with other work in the field. Finally, Section 6 wraps up the discussion and points out a set of additional research topics that must be addressed to further realize the MAGNET vision.

2 Decision processes in a MAGNET customer agent

We focus on negotiation scenarios in which the object of the interaction is to gain agreement on the performance of a set of coordinated tasks that one of the agents desires to complete in order to maximize its own utility. We assume that self-interested agents will cooperate in such a scheme to the extent that they believe it will be profitable for them to do so. After a brief high-level overview of the MAGNET system, we focus on the decision processes that must be implemented by an agent that acts as a customer in the MAGNET environment. We intend that our agents exhibit rational economic behavior. In other words, the agent should always act to maximize the *expected utility* of its principal.

We will use an example to work through the agent's decisions. Imagine that you own a small vineyard, and that you need to get last autumn's batch of wine bottled and shipped[1]. During the peak bottling season, there is often a shortage of supplies and equipment, and your small operation must lease the equipment and bring on seasonal labor to complete the process. If the wine is to be sold immediately, then labels and cases must also be procured, and shipping resources must be booked. Experience shows that during the Christmas season, wine cases are often in short supply and shipping resources are overbooked.

2.1 Agents and their environment

Agents may fulfill one or both of two roles with respect to the overall MAGNET architecture, as shown in Figure 1. A *Customer agent* pursues its goals by formulating and presenting *Requests for Quotations* (RFQs) to *Supplier agents* through a market infrastructure [1]. An RFQ specifies a task network that includes task descriptions, a precedence network, and temporal constraints that limit task start and completion times. Customer agents attempt to satisfy their goals for the greatest expected profit, and so they will accept bids at the least net cost, where cost factors can include not only bid prices, but also goal completion time, risk factors, and possibly other factors, such as preferences for specific suppliers. More precisely, these agents are attempting to maximize the utility function of some user, as discussed in detail in [2].

A supplier agent attempts to maximize the value of the resources under its control by submitting bids in response to RFQs. A bid specifies what tasks the supplier is able to undertake, when it is available to perform those tasks, how long they will take to complete, and a price. Each bid may specify one or more tasks. Suppliers may submit multiple bids to specify different combinations of tasks, or possibly different time constraints with different prices. For example, a supplier might specify a short duration for some task that requires use of high cost overtime labor, as well as a longer duration at a lower cost using straight-time labor. MAGNET currently supports simple disjunction semantics for bids from the same supplier. This means that if a supplier submits multiple bids, any non-conflicting subset can be accepted. Other bid semantics are possible [3, 4].

[1] This example is taken from the operations of the Weingut W. Ketter winery, Kröv, Germany.

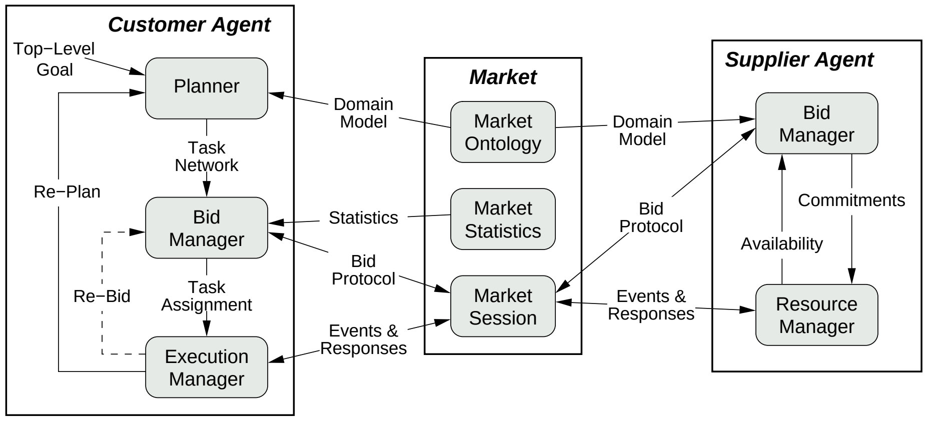

Fig. 1. The MAGNET architecture

2.2 Planning

A transaction (or possibly a series of transactions) starts when the agent or its principal acquires a goal that must be satisfied, or an opportunity arises that, if satisfied, would likely yield a positive payoff. Attributes of the goal might include a payoff and a deadline, or a payoff function that varies over time, either according to a discount rate or some other function.

While it would certainly be possible to integrate a general-purpose planning capability into a MAGNET agent, we expect that in many realistic situations the principal will already have a plan, perhaps based on standard industry practices. Figure 2 shows such a plan, for our winery bottling operation. We shall use this plan to illustrate the decision processes the agent must perform (or provide assistance to its principal in performing).

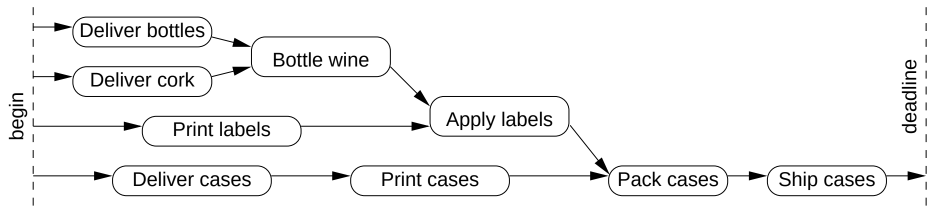

Fig. 2. Plan for the wine-bottling example.

Formally, we define a plan $\mathcal{P} = (\mathcal{S}, \mathcal{V})$ as a task network containing a set of tasks $\mathcal{S}$, and a set of precedence relations $\mathcal{V}$. A precedence relation relates two tasks $s, s' \in \mathcal{S}$ as $s \prec s'$, interpreted as "task s must be completed before task s' can start."

We assume that markets will be sponsored by trade associations and commercial entities, and will therefore be more or less specialized. A consequence of this is that agents must in general deal in multiple markets to accomplish their goals. For our example, we assume that the tasks in our plan are associated with markets as specified in Table 1.

Table 1. Tasks and market associations for the wine-bottling example

Task	Description	Market
s_1	Deliver bottles	Vineyard Services
s_2	Deliver cork	Vineyard Services
s_3	Bottle wine	Vineyard Services
s_4	Print labels	Printing & Graphic Arts
s_5	Apply labels	Vineyard Services
s_6	Print cases	Vineyard Services
s_7	Deliver cases	Vineyard Services
s_8	Pack cases	(none)
s_9	Ship cases	Transport Services

It appears that we will need to deal with 3 different markets, and we will pack the cases ourselves. Or perhaps we'll open a few bottles and invite the village to help out.

So far, our plan is not situated in time, and we have not discussed our expected payoff for completing this plan. In the wine business, the quality of the product depends strongly on time. The wine must be removed from the casks within a 2-week window, and the bottling must be done immediately. For some varieties, the price we can get for our wine is higher if we can ship earlier, given a certain quality level. All the small vineyards in the region are on roughly the same schedule, so competition for resources during the prime bottling period can be intense. Without specifying the exact functions, we assume that the payoff drops off dramatically if we miss the 2-week bottling window, and less dramatically as the shipment date recedes into the future.

This example is admittedly a bid contrived, and it is important not to stretch it too far. We are treating the bottling and labeling operations as atomic – the entire bottling operation must be finished before we can start labeling – even though common-sense would inform us that you would probably want to apply this constraint at the per-bottle level, not the per-batch level. On the other hand, some varieties of wine are aged in the bottles for 6 months or more before the labels are applied.

2.3 Planning the bidding process

At this point, the agent has a plan, and it knows which markets it must deal in to complete the plan, the value of completing the plan, and how that value depends on time. The next step is to decide how best to use the markets to maximize its utility. It will do this in two phases. First, the agent generates an overall plan for the bidding process, which may involve multiple RFQs in each of multiple markets. We call this a "bid-process plan". Then a detailed timeline is generated for each RFQ.

The simplest bid-process plan would be to issue a single RFQ in each market, each consisting of the portion of the plan that is relevant to its respective market. If all RFQs are issued simultaneously, and if they are all on the same timeline, then we can combine their bids and solve the combined winner-determination problem in a single step. However, this might not be the optimum strategy. For example:

– We may not have space available to store the cases if we are not ready to pack them when they arrive.
– Our labor costs might be much lower if we can label as we bottle; otherwise, we will need to move the bottles into storage as we bottle, then take them back out to label them.
– Once cases are packed, it is easy for us to store them for a short period. This means that we can allow some slack between the packing and shipping tasks.
– There is a limit to what we are willing to pay to bottle our wine, and there is a limit to the premium we are willing to pay to have the bottling completed earlier.

The agent can represent these issues as additional constraints on the plan, or in some cases as alternative plan components. For example, we could constrain the interval between s_5 (labeling) and s_8 (packing) to a maximum of one day, or we could add an additional storage task between s_3 (bottling) and s_5 that must be performed just in case there is a non-zero delay between the end of s_3 and the start of s_5.

There are many possible alternative actions that the agent can take to deal with these issues. It need not issue RFQs in all markets simultaneously. It need not include all tasks for a given market in a single RFQ. Indeed, dividing the plan into multiple RFQs can be an important way to reduce scheduling uncertainty. For example, we might want to have a firm completion date for the bottling and labeling steps before we order the cases.

Market statistics can be used to support these decisions. For example, if we knew that resources were readily available for the steps up through the labeling process (tasks $s_1 \ldots s_5$), we could include the case delivery and printing steps (tasks s_6 and s_7) in the same RFQ. This could be advantageous if suppliers were more likely to bid or likely to bid lower prices if they could bid on more of the business in a single interaction. In other words, some suppliers might be willing to offer a discount if we agree to purchase both bottles and cases from them, but if we negotiate these two steps in separate RFQs, we eliminate the ability to find out about such discounts.

We should note that suppliers can either help or hinder the customer in this process, depending on the supplier's motivations. For example, the supplier can help the customer mitigate issues like the constraint between bottling and packing. For example, if a supplier knew about this constraint, it could offer both tasks at appropriate times, or it could give the customer the needed scheduling flexibility by offering the case delivery over a broad time window or with multiple bids with a range of time windows. In some domains this could result in higher costs, due to the large speculative resource reservations the supplier would have to commit to in order to support its bids. On the other hand, if a supplier saw an RFQ consisting of s_6 and s_7, it would know that the customer had likely already made commitments for the earlier tasks, since nobody wants cases printed if they aren't bottling. If the supplier also knew that there would be little competition within the customer's specified time window, it could inflate its prices, knowing that the customer would have little choice.

The bid-process plan that results from this decision process is a network of negotiation tasks and decision points. Figure 3 shows a possible bid-process plan for our wine-bottling example.

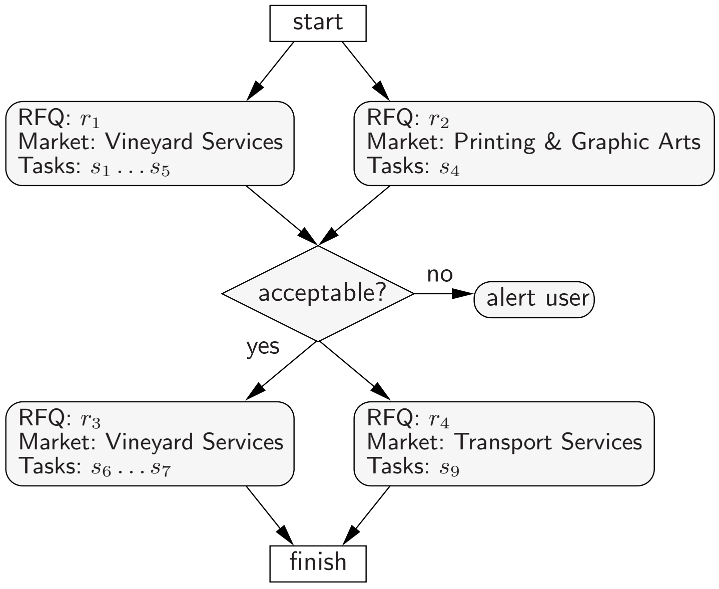

Fig. 3. Bid-process plan for the wine-bottling example.

Once we have a bid-process plan, we know what markets we will enter, and how we want to divide up the bidding process. We must then schedule the bid-process plan, and allocate time within each RFQ/bidding interaction. These two scheduling problems may need to be solved together if the bid-process plan contains multiple steps and it is important to finish it in minimum time. Each RFQ step needs to start at a particular time, or when a particular event occurs or some condition becomes true. For example, if the rules of the market require deposits to be paid when bids are awarded, the customer may be motivated to move RFQ steps as late as possible, other factors being equal. On the other hand, if resources such as our bottling and labeling steps are expected to be in short supply, the agent may wish to gain commitments for them as early as possible in order to optimize its own schedule and payoff. We assume these decisions can be supported by market statistics, the agent's own experience, and/or the agent's principal.

Each RFQ must also be allocated enough time to cover the necessary deliberation processes on both the customer and supplier sides. Some of these processes may be automated, and some may involve user interaction. The timeline in Figure 4 shows an abstract view of the progress of a single negotiation. At the beginning of the process, the customer agent must allocate deliberation time to itself to compose its RFQ[2], to the supplier for bid preparation, and to itself again for the bid evaluation process. Two of these time points, the bid deadline and the bid award deadline, must be communicated 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.

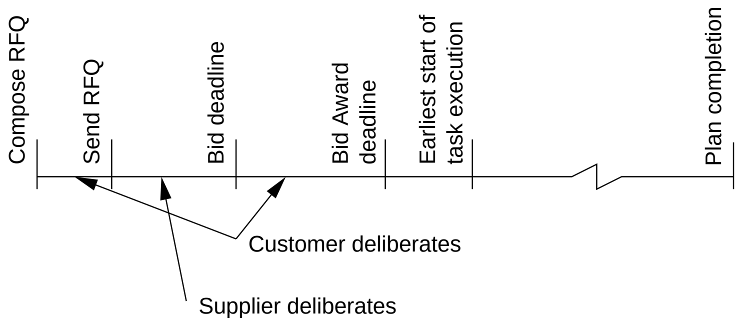

Fig. 4. Typical timeline for a single RFQ

In general, it is expected that bid prices will be lower if suppliers have more time to prepare bids, and more time and 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, and would therefore be expected to reduce bid prices. On the other hand, the customer's ability to find a good set of bids is dependent on the time allocated to bid evaluation, and if a user is making the final decision on bid awards, she may want to run multiple bid-evaluation cycles with some additional think time. We are interested in the performance of the winner determination process precisely because it takes place

[2] This may be a significant combinatorial problem – see for example [5].

within a window of time that must be determined ahead of time, before bids are received, and because we expect better overall results, in terms of maximizing the agent's utility, 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 can create opportunities for strategic manipulation of the customer by the suppliers, as discussed in [6].

The process for setting these time intervals could be handled as a non-linear optimization problem, although it may be necessary to settle for an approximation. This could consist of estimating the minimum time required for the customer's processes, and allocating the remainder of the available time to the suppliers, up to some reasonable limit.

2.4 Composing a request for quotes

At this point in the agent's decision process, we have the information needed to compose one or more RFQs, we know when to submit them, and we presumably know what to do if they fail (if we fail to receive a bid set that covers all the task in the RFQ, for example). The next step is to set the time windows for tasks in the individual RFQs, and submit them to their respective markets.

Formally, an RFQ $r = (\mathcal{S}_r, \mathcal{V}_r, \mathcal{W}_r, \tau)$ contains a subset $\mathcal{S}_r$ of the tasks in the task network $\mathcal{P}$, with their precedence relations $\mathcal{V}_r$, the task time windows $\mathcal{W}_r$ specifying constraints on when each task may be started and completed, and the RFQ timeline τ containing at least the bid deadline and bid award deadline. 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 $\mathcal{V}_r = \{\text{for each } s, s' \in \mathcal{S}_r, s' \prec s\}$, associating s with each of the other tasks $s' \in \mathcal{S}_r$ whose completion must precede the start of s.

The principal outcome of the RFQ-generation process is a set of values for the early-start and late-finish times for the time windows $\mathcal{W}_r$ in the RFQ. We obtain a crude first approximation using the Critical Path (CPM) algorithm [7], after making some assumptions about the durations of tasks, and about the earliest start time for tasks that have no predecessors in the RFQ (the *root tasks* $\mathcal{S}_R$) and the latest finish times for tasks that have no successors in the RFQ (the *leaf tasks* $\mathcal{S}_L$). Market mean-duration statistics can be used for the task durations. Overall start and finish times for the tasks in the RFQ may come from the bid-process plan, or we may already have commitments that constrain them as a result of other activities. For this discussion, we assume a continuous-time domain, although we realize that many real domains effectively work on a discrete-time basis. Indeed, it is very likely that some of our wine bottling activities would typically be quoted in whole-day increments. We also ignore calendar issues such as overtime/straight time, weekends, holidays, time zones, etc.

The critical path algorithm walks the directed graph of tasks and precedence constraints, forward from the early-start times of the root tasks to compute the earliest start $t_{es}(s)$ and finish $t_{ef}(s)$ times for each task $s \in \mathcal{S}_r$, and then backward from the late-finish times of the leaf tasks 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 specified by the RFQ, defined as $\max_{s' \in \mathcal{S}_L}(t_{ef}(s')) - \min_{s \in \mathcal{S}_R}(t_{es}(s))$, is called the *makespan* of the task network. The smallest slack in any leaf task $min_{s \in \mathcal{S}_L}(t_{lf}(s) - t_{ef}(s))$ is called the *total slack* of the task network within the RFQ. All tasks s for which $t_{lf}(s) - t_{ef}(s) = \textit{total-slack}$ are called *critical* tasks. Paths in the graph through critical tasks are called *critical paths*.

Some situations will be more complex than this. This can happen when there are constraints that are not captured in the precedence network of the RFQ. For example, some non-leaf task may have successors that are already committed but are outside the RFQ. The CPM algorithm is still applicable, but the definition of critical tasks and critical paths becomes more complex.

Figure 5 shows the result of running the CPM algorithm on the tasks of RFQ r_1 from our bid-process plan. We are assuming task durations as given in the individual "task boxes." We observe several problems immediately. The most obvious is that it is likely that many bids returned in response to this RFQ would conflict with one another because they would fail to combine feasibly. For example, if I had a bid for the label printing task s_5 for days 5-7, then the only bids I could accept for the labeling task s_4 would be those that had a late start time at least as late as day 7. If the bids for s_4 were evenly distributed across the indicated time windows, and if all of them specified the same 4-day duration, then only 1/3 of those bids could be considered. In general, we want to allow time windows to overlap, but excessive overlap is almost certainly counterproductive. We will revisit this issue shortly.

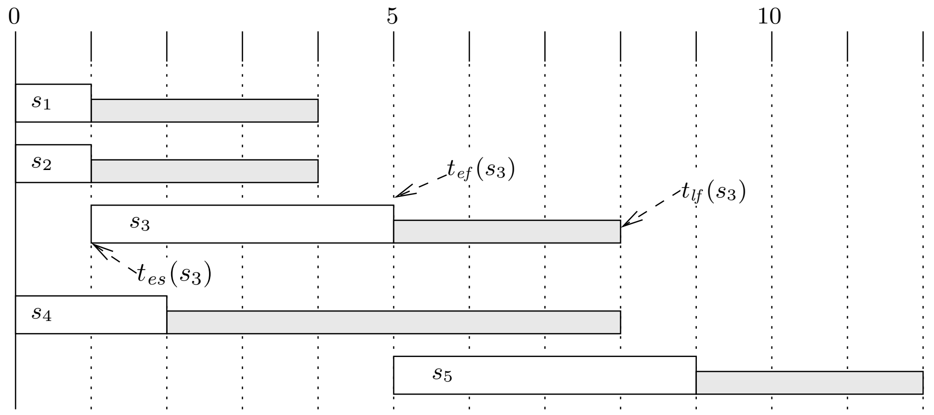

Fig. 5. Initial time allocations for tasks in RFQ r_1. Only the $t_{es}(s)$ and $t_{lf}(s)$ times are actually specified in the RFQ.

Once we have initial estimates from the CPM algorithm, there are several issues to be resolved, as described in the following sections.

Setting the total slack The plan may have a hard deadline, which may be set by a user or determined by existing commitments for tasks that cannot be started until tasks in the current RFQ are complete. Otherwise, in the normal case, the bid-process plan is expected to set the time limits for the RFQ.

It would be interesting to find a way to use the market to dynamically derive a schedule that maximizes the customer's payoff. This would require cooperation of bidders, and could be quite costly. Parkes and Ungar [8] have done something like this in a restricted domain, but it's hard to see how to apply it to the more generalized MAGNET domain.

Task ordering For any pair of tasks in the plan that could potentially be executed in parallel, we may have a choice of handling them in parallel, or in either sequential order. For example, in our wine-bottling example, we could choose to acquire the bottles before buying the corks. This example is a bit contrived, perhaps, but if there is uncertainty over the ability to complete tasks which could cause the plan to be abandoned, then (given some straightforward assumptions such as payments being due when work is completed) the agent's financial exposure can be affected by task ordering. If a risky task is scheduled ahead of a "safe" task, then if the risky task fails we can abandon the plan without having to pay for the safe task. Babanov [5] has worked out in detail how to use task completion probabilities and discount rates in an expected-utility framework to maximize the probabilistic "certain payoff" for an agent with a given risk-aversion coefficient.

For some tasks, linearizing the schedule will extend the plan's makespan, and this must be taken into account in terms of changes to the ultimate payoff. Note that in many cases the agent may have flexibility in both the start time and the completion time of the schedule. This would presumably be true of our wine-bottling example.

Allocating time to individual tasks Once we have made decisions about the overall time available and about task ordering, the CPM algorithm gives us a set of preliminary time windows. In most cases, this will not produce the best results, for several reasons:

Resource availability – In most markets, services will vary in terms of availability and resource requirements. There may be only a few dozen portable bottling and labeling machines in the region, while corks may be stored in a warehouse ready for shipping. There is a high probability that one could receive several bids for delivery of corks on one specific day, but a much lower probability that one could find even one bid for a 6-day bottling job for a specific 6-day period. More likely one would have to allow some flexibility in the timing of the bottling operation in order to receive usable bids.

Lead-time effects – In many industries, suppliers have resources on the payroll that must be paid for whether their services are sold or not. In these cases, suppliers will typically attempt to "book" commitments for their resources into the future. In our example, the chances of finding a print shop to produce our labels tomorrow is probably much lower than the chances of finding shops to print them next month. This means that, at least for some types of services, one must allow more scheduling flexibility to attract short lead time bids than for longer lead times. We should also expect to pay more for shorter lead times.

Task-duration variability – Some services are very standardized (delivering corks, printing 5000 labels), while others may be highly variable, either because they rely on human creativity (software development) or the weather (bridge construction), or because different suppliers use different processes, different equipment, or different staffing levels (wine bottling). These two types of variability can usually be differentiated by the level of predictability; suppliers that uses a predictable process with variable staffing levels are likely to be able to deliver on time on a regular basis, while services that are inherently unpredictable will tend to exhibit frequent deviations from the predictions specified in bids[3].

For services that exhibit a high variability in duration, as specified in bids, the customer's strategy may depend on whether a large number of bidders is expected, and whether there is a correlation between bid price and quoted task duration. If a large number of bidders is expected, then the customer may be able to allocate a below-average time window to the task, in the expectation that there will be some suppliers at the lower end of the distribution who will be able to perform within the specified window. On the other hand, if few bidders are expected, a larger than average time window may be required in order to achieve a reasonable probability of receiving at least one usable bid.

Excessive allocations to non-critical tasks – One obvious problem with the time allocations from the CPM algorithm as shown in Figure 5 is that non-critical tasks (tasks not on the critical path) are allocated too much time, causing unnecessary overlap in their time windows. All other things being equal, we are likely to be better off if RFQ time windows do not overlap, because we will have fewer infeasible bid combinations.

Trading off feasibility for flexibility In general we expect more bidders, and lower bid prices, if we offer suppliers more flexibility in scheduling their resources by specifying wider time windows. On the other hand, if we define RFQ time windows with excessive overlap, a significant proportion of bid combinations will be unusable due to schedule infeasibility. The intuition is that there will be some realistic market situations where the customer is better off allowing RFQ time windows to overlap to some degree, if we take into account price, plan completion time, and probability of successful plan completion (which requires at minimum a set of bids that covers the task set and can be composed into

[3] Whether the market or customers would be able to observe these deviations may depend on market rules and incentives, such as whether a supplier can be paid early by delivering early.

a feasible schedule). This means that the winner-determination procedure must handle schedule infeasibilities among bids.

Figure 6 shows a possible updated set of RFQ time windows for our wine-bottling example, taking into account the factors we have discussed. We have shortened the time windows for tasks s_1 and s_2, because we believe that bottles and corks are readily available, and can be delivered when needed. There is no advantage to allowing more time for these tasks. Market data tells us that bottling services are somewhat more difficult to schedule than labeling services, and so we have specified a wider time window for task s_3 than for s_4. Our deadline is such that the value of completing the work a day or two earlier is higher than the potential loss of having to reject some conflicting bids. We also know from market data that a large fraction of suppliers of the bottling crews can also provide the labeling service, and so the risk of schedule infeasibility will be reduced if we receive bids for both bottling and labeling. Finally, there is plenty of time available for the non-critical label-printing task s_5 without needing to overlap its time window with its successor task s_4.

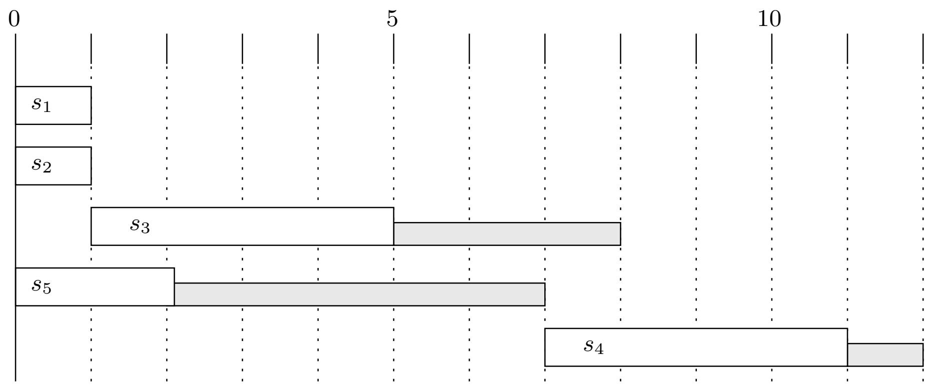

Fig. 6. Revised time allocations for tasks in RFQ r_1.

2.5 Evaluating bids

Once an RFQ is issued and the bids are returned, the agent must decide which bids to accept. The bidding process is an extended combinatorial auction, because bids can specify multiple tasks, and there are additional constraints the bids must meet (the precedence constraints) other than just covering the tasks. The winner-determination process must choose a set of bids that maximize the agent's utility, covers all tasks in the associated RFQ, and forms a feasible schedule.

Formal description of the winner-determination problem 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 $w_s \in \mathcal{W}_b$ specifies for a task s an earliest start time $t_{es}(s, b)$, a latest start time $t_{ls}(s, b)$, 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 $(t_{ls}(s, b) + d(s, b)) \leq t_{lf}(s, r)$ 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. For bids that specify multiple tasks, it is also a requirement that the time windows in the bids be internally feasible. In other words, for any bid b, if for any two of its tasks $(s_i, s_j) \in \mathcal{S}_b$ there is a precedence relation $s_i \prec s_j$ specified in the RFQ, then it is required that $t_{es}(s_i, b) + d(s_i, b) \leq t_{ls}(s_j, b)$.

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 7 shows a very small example of the problem the bid evaluator must solve. As noted before, there is scant availability of bottling equipment and crews, so we have provided an ample time window for that activity. At the same time, we have allowed some overlap between the bottling and labeling tasks, perhaps because we believed this would attract a large number of bidders with a wide variation in lead times and lower prices. Bid 1 indicates this bottling service is available from day 3 through day 7 only, and will take the full 5 days, but the price is very good. Similarly, bid 2 offers labeling from day 7 through day 10 only, again for a good price. Unfortunately, we can't use these two bids together because of the schedule infeasibility between them. Bid 3 offers bottling for any 3-day period from day 2 through day 7, at a higher price. We can use this bid with bid 2 if we start on day 4, but if we start earlier we will have to handle the unlabeled bottles somehow. Finally, bid 4 offers both the bottling and labeling services, but the price is higher and we would finish a day later than if we accepted bids 2 and 3.

Evaluation criteria We have discussed the winner-determination problem in terms of price, task coverage, and schedule feasibility. In many situations, there are other factors that can be at least as important as price. For example, we might know (although the agent might not know) that the bottling machine being offered in bid 3 is prone to breakdown, or that it tends to spill a lot of wine. We might have a long-term contract with one of the suppliers, Hermann, that gives us a good price on fertilizer only if we buy a certain quantity of corks from him every year. We might also know that one of the local printers tends to miss his time estimates on a regular basis, but his prices are often worth the

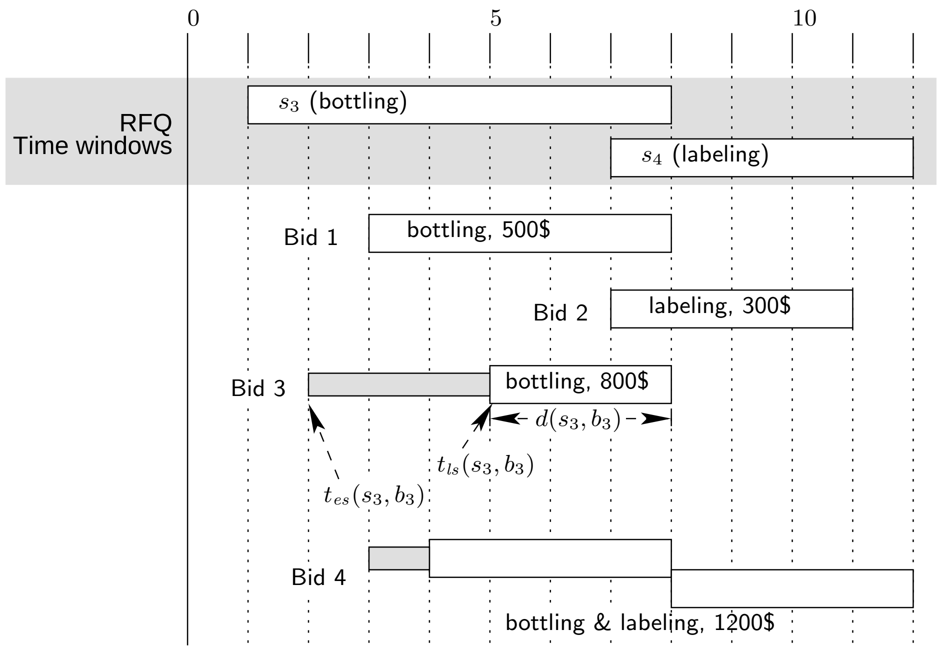

Fig. 7. Bid Example

hassle, as long as we build some slack into the schedule when we award a bid to him.

Many of these factors can be expressed as additional constraints on the winner-determination problem, and some can be expressed as cost factors. These constraints can be as simple as “don’t use bid b_3” or more complex, as in “if Hermann bids on corks, and if a solution using his bid is no more than 10% more costly than a solution without his bid, then award the bid to Hermann.” Some of them can be handled by preprocessing, some must be handled within the winner-determination process, and some will require running it twice and comparing results.

Mixed-initiative approaches There are many environments in which an automated agent is unlikely to be given the authority to make unsupervised commitments on behalf a person or organization. In these situations, we expect that many of the decision processes we discuss here will be used as decision-support tools for a human decision-maker, rather than as elements of a completely autonomous agent. The decision to award bids is one that directly creates commitment, and so it is a prime candidate for user interaction. We have constructed an early prototype of such an interface. It allows a user to view bids, add simple bid inclusion and exclusion constraints, and run one of the winner-determination search methods. Bids may be graphically overlaid on the RFQ, and both the

RFQ and bid time windows are displayed in contrasting colors on a Gantt-chart display.

Effective interactive use of the bid-evaluation functions of an agent require the ability to visualize the plan and bids, to visualize bids in groups with constraint violations highlighted, and to add and update constraints. The winner-determination solver must be accessible and its results presented in an understandable way, and there must be a capability to generate multiple alternative solutions and compare them.

2.6 Awarding bids

The result of the winner-determination process is a (possibly empty) mapping $\mathcal{S} \rightarrow \mathcal{B}$ of tasks to bids. We assume that the bids in this mapping meet the criteria of the winner-determination process: they cover the tasks in the RFQ and can be composed into a feasible schedule, and they maximize the agent's or user's expected utility. However, we cannot just award the winning bids. In general, a bid b contains one or more offers of services for tasks s, each with a duration $d(s, b)$ within a time window $w(s, b) > d(s, b)$. The price assumes that the customer will specify, as part of the bid award, a specific start time for each activity. Otherwise, the supplier would have to maintain its resource reservation until some indefinite future time when the customer would specify a start time. This would create a disincentive for suppliers to specify large time windows, raise prices, and complicate the customer's scheduling problem.

This means that the customer must build a final work schedule before awarding bids. We will defer to the next section the issue of dealing with schedule changes as work progresses. This scheduling activity represents another opportunity to maximize the customer's expected utility. In general, the customer's utility at this point is maximized by appropriate distribution of slack in the schedule, and possibly also by deferring task execution in order to defer payment for completion.

3 Solving the MAGNET winner-determination problem

We now focus on the MAGNET winner-determination problem, originally introduced in Section 2.5. Earlier we have described both an Integer Programming formulation [9] and a simulated annealing framework for solving this problem [10] for this problem. Here we will focus on an application of the A* method. The algorithm presented here solves the winner-determination problem under assumption of a fixed payoff, and does not deal with a payoff that depends on completion time.

The winner determination problem for combinatorial auctions has been shown to be $\mathcal{NP}$-complete and inapproximable [11]. This result clearly applies to the MAGNET winner determination problem, since we simply apply an additional set of (temporal) constraints to the basic combinatorial auction problem, and we don't allow free disposal. In fact, because the additional constraints create additional bid-to-bid dependencies, and because bids can vary in both price and in time specifications, the bid-domination and partitioning methods used by others to simplify the problem (for example, see [12]) cannot be applied in the MAGNET case.

Sandholm has shown that there can be no polynomial-time solution, nor even a polynomial-time bounded approximation [12], so we must accept exponential complexity. We will see in Section 4 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.

Sandholm described an approach to solving the standard combinatorial auction winner-determination problem [12] using an iterative-deepening A* formulation. Although many of his optimizations, such as the elimination of dominated bids and partitioning of the problem, cannot be easily applied to the MAGNET problem, we have adapted the basic structure of Sandholm's formulation, and we have improved upon it by specifying a means to minimize the mean branching factor in the generated search tree.

In general, tree search methods are useful when the problem at hand can be characterized by a solution path in a tree that starts at an initial node (root) and progresses through a series of expansions to a final node that meets the solution criteria. Each expansion generates successors (children) of some existing node, expansions continuing until a final node is found. The questions of which node is chosen for expansion, and how the search tree is represented, lead to many different search methods. In the A* method, the node chosen for expansion is the one with the "best" evaluation[4], and the search tree is typically kept in memory in the form of a sorted queue. A* uses an evaluation function

$$f(N) = g(N) + h(N)$$

for a node N, where $g(N)$ is the cost of the path from initial node N_0 to node N, and $h(N)$ is an estimate of the remaining cost to a solution node. If $h(N)$ is a strict lower bound on the remaining cost (upper bound for a maximization problem), we call it an *admissible heuristic* and A* is complete and optimal; that is, it is guaranteed to find a solution with the lowest evaluation, if any solutions exist, and it is guaranteed to terminate eventually if no solutions exist.

We describe a basic A* formulation of the MAGNET winner-determination problem, and then we show how this formulation can be adapted to a depth-first iterative-deepening model [13] to reduce or eliminate memory limitations.

3.1 Bidtree framework

For a basic introduction to the A* algorithm, see [14], or another textbook on Artificial Intelligence. Our formulation depends on two structures which must be prepared before the search can run. The first is the *bidtree* introduced by

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

Sandholm, and the second is the *bid-bucket*, a container for the set of bids that cover the same task set.

A bidtree is a binary tree that allows lookup of bids based on item content. The bidtree is used to determine the order in which bids are considered during the search, and to ensure that each bid combination is tested at most once. In Sandholm's formulation, the collection of bids into groups that cover the same item sets supports the discard of dominated bids, with the result that each leaf in the bidtree contains one bid. However, because our precedence constraints create dependencies among bids in different buckets, bid domination is a much more complex issue in the MAGNET problem domain. Therefore, we use bid-buckets at the leaves rather than individual bids.

The principal purpose of the bidtree is to support content-based lookup of bids. Suppose we have a plan $\mathcal{S}$ with tasks $s_m, m = 1..4$. Further suppose that we have received a set of bids $b_n, n = 1..10$, with the following contents: $b_1 : \{s_1, s_2\}$, $b_2 : \{s_2, s_3\}$, $b_3 : \{s_1, s_4\}$, $b_4 : \{s_3, s_4\}$, $b_5 : \{s_2\}$, $b_6 : \{s_1, s_2, s_4\}$, $b_7 : \{s_4\}$, $b_8 : \{s_2, s_4\}$, $b_9 : \{s_1, s_2\}$, $b_{10} : \{s_2, s_4\}$. Figure 8 shows a bidtree we might construct for this problem. Each node corresponds to a task. One branch, labeled *in*, leads to bids that include the task, and the other branch, labeled *out*, leads to bids that do not.

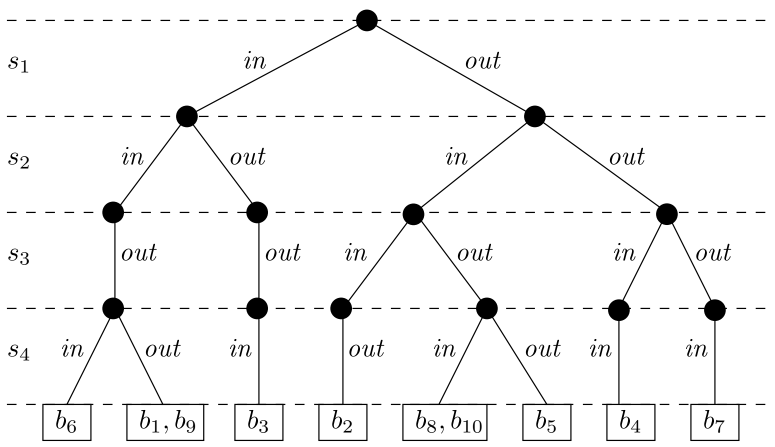

Fig. 8. Example bidtree, lexical task order

We use the bidtree by querying it for bid-buckets. A query consists of a *mask*, a vector of values whose successive entries correspond to the "levels" in the bidtree. Each entry in the vector may take on one of three values, $\{$*in*, *out*, *any*$\}$. A query is processed by walking the bidtree from its root as we traverse the vector. If an entry in the mask vector is *in*, then the *in* branch is taken at the corresponding level of the tree, similarly with *out*. If an entry is *any*, then both branches are taken at the corresponding level of the bidtree. So, for example, if we used a mask of [*in*, *any*, *any*, *in*], the bidtree in Figure 8 would return the bid-buckets containing $\{b_6\}$ and $\{b_3\}$.

A bid-bucket is a container for a set of bids that cover the same task set. In addition to the bid set, the bid-bucket structure stores the list of other bid-buckets whose bids conflict with its own (where we use "conflicts" to mean that they cover overlapping task sets). This recognizes the fact that all bids with the same task set will have the same conflict set.

In order to support computation of the heuristic function, we use a somewhat different problem formulation for A* and IDA* than we used for the IP formulation described in [9]. In that formulation, we were minimizing the sum of the costs of the selected bids. In this formulation, we minimize the cost of each of the tasks, given a set of bid assignments. This allows for straightforward computation of the A* heuristic function $f(N)$ for a given node N in the search tree. We first define

$$f(N) = g(\mathcal{S}_m(N)) + h(\mathcal{S}_u(N))$$

where $\mathcal{S}_m(N)$ is the set of tasks that are mapped to bids in node N, while $\mathcal{S}_u(N) = \mathcal{S}_r \setminus \mathcal{S}_m(N)$ is the set of tasks that are not mapped to any bids in the same node. We then define

$$g(\mathcal{S}_m(N)) = \sum_{j|s_j \in \mathcal{S}_m} \frac{c(b_j)}{n(b_j)}$$

where b_j is the bid mapped to task s_j, $c(b_j)$ is the total cost of b_j, $n(b_j)$ is the number of tasks in b_j, and

$$h(\mathcal{S}_u(N)) = \sum_{j|s_j \in \mathcal{S}_u} \frac{c(b_j^\star)}{n(b_j^\star)}$$

where $b_j^\star$ is the "usable" bid for task s_j that has the lowest cost/task. By "usable," we mean that the bid $b_j^\star$ includes s_j, and does not conflict (in the sense of having overlapping task sets) with any of the bids b_j already mapped in node N.

Note that, unlike the case with the IP solver, the definition of $g(\mathcal{S}_m(N))$ can be expanded to include other factors, such as risk estimates or penalties for inadequate slack in the schedule, and these factors can be non-linear. The only requirement is that any such additional factor must *increase* the value of $g(\mathcal{S}_m(N))$, and not decrease it, because otherwise the admissibility of the heuristic will be compromised, and we no longer would have a complete search method.

3.2 A* formulation

Now that we have described the bidtree and bid-bucket, we can explain our optimal tree search formulation. The algorithm is given in Figure 9.

The principal difference between this formulation and the "standard" A* search formulation (see, for example, [14]), is that nodes are left on the queue

```
1  Procedure A*_search
2  Inputs:
3      $\{\mathcal{S}, \mathcal{V}\}$: the task network to be assigned
4      $\mathcal{B}$: the set of bids, represented as a bidtree
5  Output:
6      $N_{opt}$: the node having a mapping $\mathcal{M}(N_{opt}) = \mathcal{S} \rightarrow \mathcal{B}$
              of tasks to bids with an optimal evaluation, if one exists
7  Process:
8      $Q \leftarrow$ priority_queue, sorted by node evaluation $f(N)$
9      $N_0 \leftarrow$ empty node
10     $mask \leftarrow \{in, any, any, cdots\}$
11     $\mathcal{B}^c_{N_0} \leftarrow$ bidtree_query($\mathcal{B}, mask$)
           "$\mathcal{B}^c_{N_0}$ is a set of bids (in the form of a set of bid buckets), containing
           the bids that can be used to expand $N_0$"
12     insert($Q, N_0$)
13     loop
14         if empty($Q$) then return failure
15         $N \leftarrow$ first($Q$)
16         if solution($N$) then return $N$
17         $N' \leftarrow$ astar_expand($N$) "see Figure 10"
18         if $N' = null$ then remove_front($Q$) "Remove nodes that fail to expand"
19         else if feasible($N'$) then insert($Q, N'$)
```

Fig. 9. Bidtree-based A* search algorithm.

(line 15) until they cannot be expanded further, and only a single expansion is tried (line 17) at each iteration. This is to avoid expending unnecessary effort evaluating nodes.

The expansion of a parent node N to produce a child node N' (line 17 in Figure 9) using the bidtree is shown in Figure 10. Here we see the reason to keep track of the buckets for the candidate-bid set of a node. In line 16, we use the mask for a new node to retrieve a set of bid-buckets. In line 18, we see that if the result is empty, or if there is some unallocated task for which no usable bid remains, we can go back to the parent node and just dump the whole bucket that contains the candidate we are testing.

In line 17 of Figure 10, we must find the minimum-cost "usable" bids for all unallocated tasks $\mathcal{S}_u$ (tasks not in the union of the task sets of $\mathcal{B}_{N'}$), as discussed earlier. One way (not necessarily the most efficient way) to find the set of usable bids is to query the bidtree using the mask that was generated in line 14, changing the single *in* entry to *any*. If there is any unallocated task that is not covered by some bid in the resulting set, then we can discard node N' because it cannot lead to a solution (line 22). Because all other bids in the same bidtree leaf node with the candidate bid b_x will produce the same bidtree mask and the same usable-bid set, we can also discard all other bids in that leaf node from the candidate set of the parent node N.

```
1  Procedure astar_expand
2  Inputs:
3      $N$: the node to be expanded
4  Output:
5      $N'$: a new node with exactly one additional bid, or null
6  Process:
7      $buckets \leftarrow \emptyset$
8      while $buckets = \emptyset$ do
9          if $\mathcal{B}_N^c = \emptyset$ then return $null$ "$\mathcal{B}_N^c$ is set of candidate bids for node $N$"
10         $b_x \leftarrow$ choose($\mathcal{B}_N^c$) "pick a bid from the set of candidates"
11         $\mathcal{B}_N^c \leftarrow \mathcal{B}_N^c - b_x$ "remove the chosen bid from the set"
12         $N' \leftarrow$ new node
13         $\mathcal{B}_{N'} \leftarrow \mathcal{B}_N + b_x$ "$\mathcal{B}_{N'}$ is the set of bids in node $N'$"
14         $\mathcal{S}_u \leftarrow$ unallocated_tasks($N'$) "tasks not covered by any bid $b \in \mathcal{B}_{N'}$"
15         $mask \leftarrow$ create_mask($\mathcal{B}'_N$)
               "for each task that is covered by a bid in $\mathcal{B}_{N'}$, set the corresponding
               entry to out. Then find the first task in $s \in \mathcal{S}_u$ (the task in $\mathcal{S}_u$ with
               the minimum index in the bidtree) and set its entry to in. Set the
               remaining entries to any"
16         $buckets \leftarrow$ bidtree_query($\mathcal{B}$, $mask$)
17         $\mathcal{B}_u \leftarrow \forall s \in \mathcal{S}_u$, minimum_usable_bid($s$) "see the narrative"
18         if (solution($N'$))
               $\vee((buckets \neq \emptyset) \wedge (\neg\exists s \in \mathcal{S}_u |$minimum_usable_bid$(s) = null))$
19         then
20             $\mathcal{B}_{N'}^c \leftarrow buckets$ "candidates for $N'$"
21         else
22             remove($\mathcal{B}_N^c$, bucket($b_x$))
                   "all bids in the bucket containing $b_x$ in node $N$ will produce the
                   same mask and therefore an empty candidate set or a task that
                   cannot be covered by any usable bid"
23     end while
24     $g(N') \leftarrow \sum_{b \in \mathcal{B}_{N'}} c_b$
25     $h(N') \leftarrow \sum_{b \in \mathcal{B}_u}$ avg_cost($b$)
26     return $N'$
```

Fig. 10. Bidtree-based node-expansion algorithm.

This implementation is very time-efficient but A* fails to scale to large problems because of the need to keep in the queue all nodes that have not been fully expanded. Limiting the queue length destroys the optimality and completeness guarantees. Some improvement in memory usage can be achieved by setting an upper bound once the first solution is found in line 18 of Figure 10. Once an upper bound *flimit* exists, then any node N for which $f(N) > flimit$ can be safely discarded, including nodes already on the queue. Unfortunately, this helps only on the margin; there will be a very small number of problems for which the resulting reduction in maximum queue size will be sufficient to convert a failed or incomplete search into a complete one. We address this in the next section.

One of the design decisions that must be made when implementing a bidtree-based search is how to order the tasks (or items, in the case of a standard combinatorial auction) when building the bidtree. It turns out that this decision can have a major impact on the size of the tree that must be searched, and therefore on performance and predictability. As we have shown in [15], the tasks should be ordered such that the tasks with higher numbers of bids come ahead of tasks with lower numbers of bids. This ordering is exploited in line 18 of Figure 10, where bid conflicts are detected.

3.3 Iterative Deepening A*

Iterative Deepening A* (IDA*) [13] is a variant of A* that uses the same two functions g and h in a depth-first search, and which keeps in memory only the current path from the root to a particular node. In each iteration of IDA*, search depth is limited by a threshold value *flimit* on the evaluation function $f(N)$.

```
Procedure IDA*_search
Inputs:
    $\{\mathcal{S}, \mathcal{V}\}$: the task network to be assigned
    $\mathcal{B}$: the set of bids, represented as a bidtree
Output:
    $N_{opt}$: the node having a mapping $\mathcal{M}(N_{opt}) = \mathcal{S} \rightarrow \mathcal{B}$ of tasks to bids
            with an optimal evaluation, if one exists
Process:
    $N_0 \leftarrow$ empty node
    $g(N_0) \leftarrow 0$
    $h(N_0) \leftarrow \sum_{s \in \mathcal{S}}$ avg_cost(minimum_bid($s$))
    $flimit \leftarrow f(N_0)$
    $mask \leftarrow \{in, any, any, \cdots\}$
    $best\_node \leftarrow null$
    while $(best\_node = null) \wedge (flimit \neq \infty)$ do
        $\mathcal{B}^c_{N_0} \leftarrow$ bidtree_query($\mathcal{B}, mask$)
            "$\mathcal{B}^c_{N_0}$ is a set of bids (in the form of a set of bid buckets),
            containing the bids that can be used to expand $N_0$. We have to
            repeat this for every iteration."
        $new\_limit \leftarrow$ dfs_contour($N_0$)
        if $best\_node = null$ then
            $flimit \leftarrow$ max($new\_limit, z \cdot flimit$) "see narrative in this section"
    end while
    return $best\_node$
```

Fig. 11. Bidtree-based Iterative Deepening A* search algorithm: top level.

We show in Figure 11 a version of IDA* that uses the same bidtree and node structure as the A* algorithm. The recursive core of the algorithm is shown in Figure 12. This search algorithm uses the same node expansion algorithm as we used for the A* search, shown in Figure 10 above.

```
 1 Procedure dfs_contour
 2 Inputs:
 3     $N$: a node
 4 Output:
 5     $new\_limit$: a candidate for the $flimit$ value for the next contour. This is
                   either the first $f(N)$ value seen that is larger than $flimit$, or
                   the value of the best solution node found. If it is determined
                   that no solution is possible, then we return $\infty$.
 6 Process:
 7     $new\_limit \leftarrow f(N)$
 8     if $new\_limit > flimit$ then "enforce contour limit"
 9         return $new\_limit$
10     if solution($N$) then "switch to branch-and-bound"
11         $flimit \leftarrow new\_limit$  "set new upper bound"
12         $best\_node \leftarrow N$
13         return $new\_limit$
14     $nextL \leftarrow \infty$
15     while $\mathcal{B}_N^c \neq \emptyset$ do
16         $N' \leftarrow$ astar_expand($N$)
17         if $(N' \neq null) \wedge$ (feasible($N'$)) then
18             $new\_limit \leftarrow$ dfs_contour($N'$)
19             $nextL \leftarrow$ min($nextL, new\_limit$)
20     end while
21     return $nextL$
```

Fig. 12. Bidtree-based Iterative Deepening A* search algorithm: depth-first contour.

There are three issues to note in this algorithm:

- The tuning parameter z shown in line 19 of Figure 11 is a positive number > 1. This controls the amount of additional depth explored in each iteration of the main loop that starts on line 15. Experimentation shows that a good value is 1.15, and that it is not very sensitive (performance falls off noticeably with $z < 1.1$ or $z > 1.2$).
- Whenever a solution is found, the value of *flimit* is updated in line 11 of Figure 12. This follows the usage in Sandholm [12], and limits exploration to nodes (and solutions) that are better than the best found so far.
- We test nodes for feasibility in line 17 of Figure 12 to prevent consideration and further expansion of nodes that cannot possibly lead to a solution.

4 Search performance

The winner determination problem must be solved within the confines of a time-limited negotiation scenario. More significantly, the agent must allocate time to the winner-determination process when it sets the negotiation timeline prior to issuing an RFQ. Failure to solve the winner-determination problem within the allocated time will result in failure of the negotiation process.

Our goal is to measure the probability distributions of search times across a range of easily-measured (or easily-estimated) 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 have chosen four problem size and complexity metrics for evaluation.

Task count – This is simply the number of tasks m in a task network, and can be directly measured.

Bid count – The number of bids submitted n. Alternatively, the number of bids/task. We will assume that this value can be estimated from market statistics, given the task network composition and possibly other data such as lead time or allowable schedule slack.

Bid size – The mean number of tasks specified in each bid $\overline{|\mathcal{S}_i|}, i = 1 \ldots n$. We assume this can also be estimated from market statistics. We can think of a "specialist market" as being one in which most bidders bid on only one or a few task types, while a "generalist market" is one in which many bidders will bid on large chunks of a plan.

Plan complexity – The mean size of the precedence set of a task in the task network. This can be directly measured in the task network.

Some of these problem-size parameters can be controlled independently, and some are not as independent as we might like. For example, when we increase the number of tasks in the task network, we also need to increase the number of bids and/or the number of tasks/bid if we wish to retain the same number of bids for each task. For that reason, we will use bids/task and bid size/task network size when it is important to consider independent variables.

In the remainder of this chapter, we report on a set of experiments that give us the necessary probability distribution data for the IDA* winner-determination method. There are clearly limits on its scalability, because the problem has unavoidable exponential complexity. We shall observe runtime characteristics with exponential tails, so there will be unpredictable situations where good solutions will not be found within any fixed time limit. This is just another way of saying that 100% probability of finding a solution within a predetermined time limit is not achievable. Instead, our goal is to develop the data that will allow us to convert a desired probability of success to a time allocation, given the necessary problem-size metrics.

4.1 Experimental setup

The experimental setup consists of a plan generator to produce randomly-generated task networks, a bid generator to produce randomly-generated bids for each task network, and a bid evaluator to perform the winner-determination search. The winner-determination process is instrumented to measure both elapsed time and the number of steps performed.

The problem-generation process requires a stream of random numbers. In order to be able to repeat test conditions, we maintain 2 separate streams, one for generating plans and RFQs, and one for generating bids Each can be initialized with a seed, so we have the ability to repeat the same plan sets with different bid sets.

All experiments were run using a dedicated 1800 MHz Intel/Linux machine. Timings are given in wall-clock time. The MAGNET system (including the IP preprocessor and the IDA* 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/`.

Customer: Generate plan For these experiments, plans are randomly-generated task networks, with a number of controllable parameters. Task Network variables include:

Number of tasks

Mix of task types – Task types are characterized by expected duration d_e, duration variability $\sigma(d)$, average price c_e, and price variability $\sigma(c)$. Both duration and price are normally distributed, positive values (the distributions are truncated at 0). For all experiments, the task types and their properties are as given in Table 2.

Branch factor – This controls the average number of precedence relationships generated per task, which we call "fan-in." For example, if a particular task has two predecessors, then the fan-in value for that task is 2. As the plan is built, each new task s_j is linked to each of the previous tasks $s_{j'}, j' = 1 \ldots (j-1)$ with a probability p_l of $(\mathit{branchFactor}/(\mathit{taskCount}-1))$. The completed network is filtered to remove most redundant precedence relations, so the final number is lower than the initial number generated.

Table 2. Task types and relative proportions used in performance experiments

		Duration		Price		Resource
Name	Proportion	d_e	$\sigma(d)$	c_e	$\sigma(c)$	Availability
short	40%	2.0	0.4	500	0.2	0.8
medium	40%	3.0	0.2	1000	0.3	0.7
long	20%	8.0	0.3	1500	0.3	0.4

As an example, Figure 13 shows the task network for the first of the problems used in the bid-count test series described in Section 4.2 below. Keep in mind that duration values at this point in the process are expected values. Actual durations cannot be known until bids are awarded.

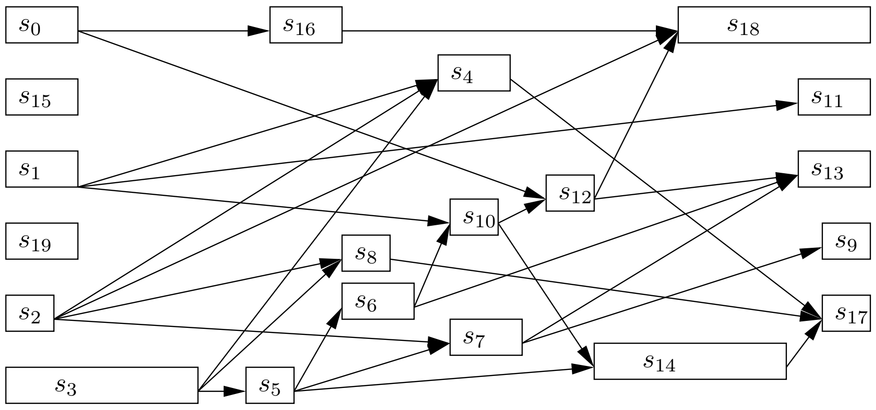

Fig. 13. Task network for Problem #1. Box widths indicate relative task durations.

Customer: Construct and issue a request for quotes The output of the planning step is simply a task network, made up of tasks for which we have some statistical data. Before we can ask for bids, we must add some constraints to communicate our desires more clearly to potential bidders, and we must consider the availability of bidders and their resources. The goal is to maximize the "usefulness" and minimize the cost of the bids we receive. Ultimately, we would expect to use Babanov's method based on Expected Utility to do this [5], but for these experiments we use a simpler approach that generates bids with well-defined statistics that are adequate to exercise our winner-determination solvers. We illustrate the RFQ generation process with the example task network shown in Figure 14.

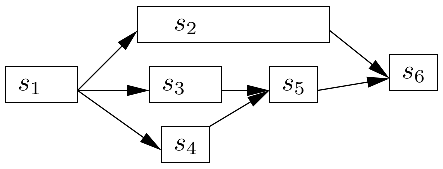

Fig. 14. Example task network for RFQ illustration.

For these experiments, the RFQ time windows $\mathcal{W}_r$ are computed as follows:

1. Compute the "expected makespan" d_m for the entire task network. The makespan is simply the sum of the expected durations of the "critical" tasks,

or the tasks that are on the critical path assuming all tasks are assigned their expected durations.

2. Add some amount of "plan slack" ζ to the task network. This simply relaxes the plan deadline by some fraction. If the start time of the plan is t_0, then the earliest possible completion time, assuming all tasks are completed within their expected durations, is $t_m = t_0 + d_m$. After applying plan slack, we have $t_{goal} = t_0 + \zeta d_m$.
3. Determine final time windows for individual tasks. This is done in two steps. The first step is to set the minimum time allocation $d(s_j)$ for each task s_j to some value $d_{\min}(s_j) = \eta d_e(s_j)$ where $\eta <= \zeta$. Then we run the CPM algorithm with a start time of t_0, and a deadline of t_{goal}. Figure 15 shows the resulting start and finish times for the task network of Figure 14, using $\zeta = 1.2$ and $\eta = 1.15$.

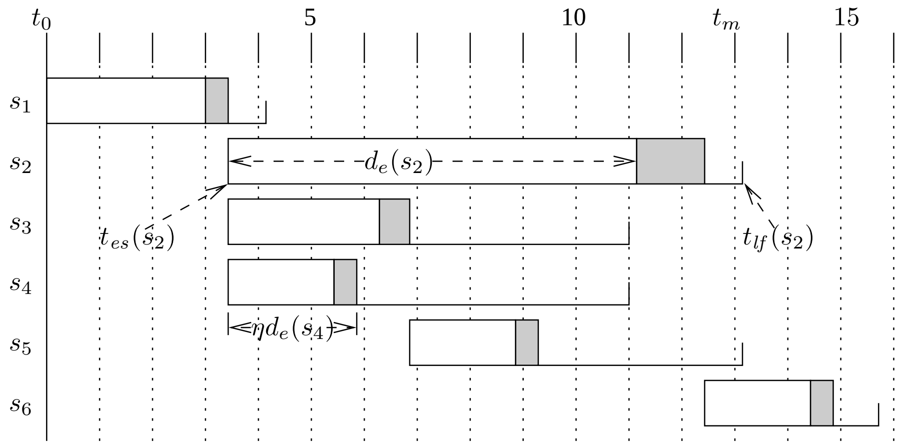

Fig. 15. Gantt chart for task network of Figure 14 with $\zeta = 1.2$, $\eta = 1.15$.

There is no doubt that this procedure generates time windows that are suboptimal, because of the large overlaps produced for noncritical tasks. However, it is very adequate for our experimental purposes, because it exercises the feasibility testing by ensuring that virtually all problems will have some infeasibilities among bids.

Supplier: Generate bids For these experiments, we used a test component that masquerades as an entire community of supplier agents. Each time a new RFQ is passed to it, it attempts to generate a specified number of bids.

For a given RFQ r, individual bids b_i are generated as follows:

1. Select a task s_j at random from $\mathcal{S}_r$, and attempt to generate bid parameters for s_j.

2. Use the task-type parameters to generate a supplier time window
$$w_j(b_i) = \{t_{es}(s_j, b_i), t_{ls}(s_j, b_i), d(s_j, b_i)\}$$
for each task. We first generate the duration $d(s_j, b_i)$ using a normal distribution with mean and standard deviation equal to the expected duration and variability from the task type. If the resulting duration
$$d(s_j, b_i) > (t_{lf}(s_j, r) - t_{es}(s_j, r))$$
then it cannot be used and the attempt is abandoned. Otherwise, an early start $t_{es}(s_j, b_i)$ is generated from a uniform distribution over the interval
$$[t_{es}(s_j, r), (t_{lf}(s_j, r) - d(s_j, b_i))] ,$$
and a late start over the remaining slack in the RFQ time window
$$[t_{es}(s_j, b_i), (t_{lf}(s_j, r) - d(s_j, b_i))] .$$
Actually, this step is slightly more complex after the first task-bid is generated, because of the need to generate bids with internal feasibility. We simply tighten up each RFQ time window with the maximum early finish of preceding tasks and the minimum late start of succeeding tasks.
3. If we successfully generate a time window, we call it a "valid task specification" and add the task s_j and its time window $w_j(b_i)$ to the bid. If a valid task specification was generated, then with probability p_{link}, each predecessor and successor link from task s_j is followed to choose additional tasks $s_{j'}$ to add to the bid, and so on recursively.
4. To complete the process for a single bid, we determine a cost for the overall bid and return it.
5. Because valid task specifications are not always achieved, some attempts to generate bids will fail, and so the number of bids actually generated is nearly always somewhat smaller than the target number. We can optionally check the resulting bid-set before it is returned, and test for coverage. If coverage is not achieved, then for each missing task, we make an additional attempt to generate a bid, with the missing task selected as a starting point in Step 1 rather than a random task. This is useful because returned bid-sets that do not cover all tasks will not exercise our winner-determination solvers. All experiments reported in this chapter used this feature.

The resulting bids specify "contiguous" sets of tasks, and each bid is guaranteed to be internally feasible.

4.2 Characterizing the Iterative Deepening A* solver

In this section we examine the performance of the bidtree-based IDA* solver that was described in Section 3. We want to examine the scalability and predictability of this algorithm, and to compare it with the IP solver.

Each problem set consists of 100 problems, with randomly-generated task networks 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, Bid Size, and Network Complexity (branch factor).

Bid count experiment Here we examine the scalability of the IDA* 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 3 represents 100 problems, each with 30 tasks and varying numbers of bids. The same set of 100 task networks is used in each row; only the bid sets are varied. In the table, the "Bid Size" column gives the average size of bids (number of tasks per bid). The "# Solved" column gives the number of problems solved out of 100 (not all 100 problems were necessarily solvable), and "Mean Time" gives the mean search time in milliseconds. The meaning of the remaining 3 columns will become clear shortly.

Table 3. Bid Count experiment for the IDA* solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Mean Time (ms)	m	v	χ^2 (9 dof)
30	80	7.45	64	45.4	15.9	1.35	2.36
30	106	7.45	82	106	48.5	1.33	1.48
30	133	7.53	99	894	163	1.96	7.33
30	159	7.55	100	2180	500	2.36	3.50
30	186	7.48	100	34,700	1410	3.03	1.18
30	213	7.38	100	22,000	3070	3.33	2.92
30	239	7.46	99	137,000	9230	4.80	0.83
30	265	7.35	100	152,000	16,000	4.55	1.60

While the data in Table 3 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 16 the complete runtime distributions for these problem sets. For each curve, we show actual observations along with a lognormal distribution that minimizes the χ^2 metric[5]. In Table 3, the last 3 columns give the parameters of the inferred lognormal density function. The m column is the median value (the value of $\log m$ is the mean of the log of the distribution), and the v column is the standard deviation of the log of the distribution. We also give χ^2 values, computed by the equiprobable method [16]. For 9 degrees of freedom, $\chi^2 = 3.5$ corresponds to roughly a 95% confidence that the data fit the hypothesized distribution. In general, a large value of χ^2 shows up in a plot such as Figure 16 as a very obvious failure of the observations to lie atop the inferred distribution curve.

It seems clear that the 186-bid row in Table 3 contains a large outlier. Figure 16 shows the observed and inferred distributions for all of the problem sets in this experiment. We can see visually that the lognormal distributions provide a good approximation for this data. For 9 dof, the 95% confidence point

[5] 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 a measure of the deviation from a "perfect fit".

is approximately 3.5, so all but one of these sets is at or above a 95% match to the given lognormal distribution. This gives us confidence that we can base predictions on these inferred distributions. The single outlier in the 186-bid set can be seen in the plot at about $3 \cdot 10^6$ msec.

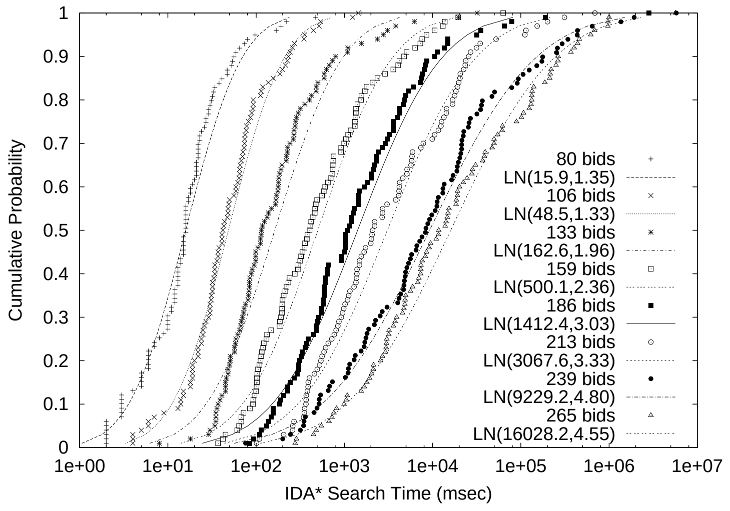

Fig. 16. IDA* run-time distributions for problems with 30 tasks, bid count ranging from 80 bids to 265 bids, bid size approx. 7.5 tasks/bid.

Bid size experiment We now turn to the bid-size dimension of problem variability. We expect that larger average bid sizes will result in shorter search times, because the number of bids in a solution will be smaller. This leads to fewer combinations in the search space, and fewer precedence constraints that must be generated and solved (since bids are required to be internally feasible). One way to see this is to look at an upper bound on the number of possible solutions as bid size varies. A simple upper bound is the number of combinations of bids $C = n!/(n-(m/b))!$ where n is the number of bids, m is the number of tasks, and b is the mean number of tasks/bid. Sandholm [12, 17] avoids this with small bid sizes by partitioning the bids into non-overlapping subsets. This works because small bids have fewer overlaps, and because the items in a simple combinatorial auction are arbitrarily separable, there being no constraints among them. We cannot use this approach because of the presence of precedence constraints that connect the items at auction (the tasks).

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. The task networks in these problem sets are the same as the ones used in the bid-count experiment in the previous section.

For this experiment, we generated 100 problems with 30 tasks and 150 "bid attempts". We then varied the probability that the bidder will follow precedence links in generating multiple-task bids from 0.3 to 0.9. Table 4 gives the raw data and the parameters for the inferred lognormal distributions for this experiment. All but the 7.53 tasks/bid and the 11.5 tasks/bid sets fit the lognormal distribution quite well, as evidenced by the χ^2 values.

Table 4. Bid Size experiment for the IDA* solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Mean Time (ms)	m	v	χ^2 (9 dof)
30	134	2.43	96	3,120,000	21,900	7.76	1.13
30	133	3.81	92	800,000	1300	5.38	1.66
30	133	5.58	97	2030	314	2.61	1.13
30	133	7.53	99	936	163	1.98	7.10
30	133	9.00	90	249	110	1.34	3.82
30	133	10.30	91	121	60.2	0.874	3.17
30	133	11.50	92	69.0	48.4	0.483	7.63

Figure 17 shows the observed and inferred runtime distributions for selected sets from this experiment. Clearly, both difficulty and variability rise significantly as bid size is reduced.

Task count experiment The next 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). We again generated 100 problems for each set, varying the task count over a 10:1 range from 5 tasks to 50 tasks, with the number of "bid attempts" at about 3.5 bids/task. Table 5 gives the raw data and the parameters for the inferred lognormal distributions for this experiment. These data do not fit the "inferred" lognormal distributions as well as the data in the previous section, for reasons that are unclear. In the first two sets that appears to be because of quantizing error (the clock resolution is only 1 ms, and the mean search time for the 5-task set was just 2.12 ms). For the other poor fits in the 35-task and 50-task sets, the errors seem unlikely to strongly impact our ability to use the 95% point as an estimator, since there is a crossover between the data and the lognormal curve in that region.

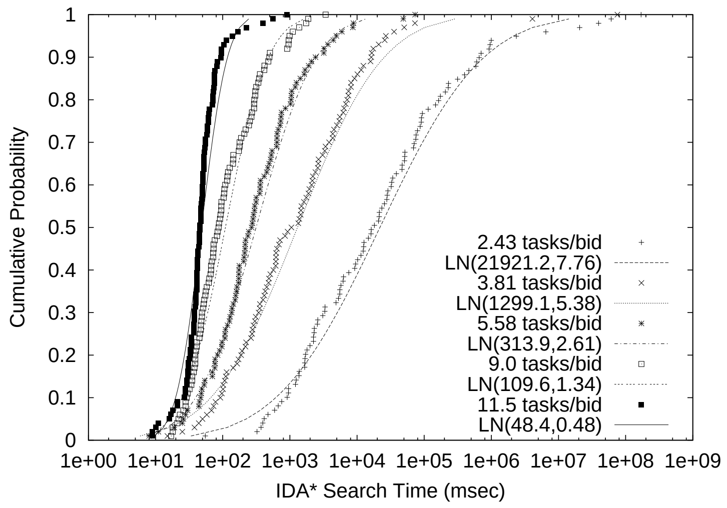

Fig. 17. IDA* run-time distributions for problems with 30 tasks, 133 bids, bid size ranging from 2.4 tasks to 11.5 tasks/bid.

Table 5. Task Count experiment for the IDA* solver

Task Count	Bid Count	Bid Size	# Solved (of 100)	Mean Time (ms)	m	v	χ^2 (9 dof)
5	13.5	2.10	94	2.12	1.81	0.499	31.7
10	29.0	3.27	94	4.78	3.01	0.716	9.27
15	45.4	4.46	90	9.22	5.93	0.664	4.02
20	61.0	5.45	85	22.4	12.2	1.18	3.97
25	77.7	6.30	85	31.1	19.0	0.874	0.76
30	93.4	7.40	80	81.4	27.3	0.955	4.03
35	111	8.42	74	914	47.0	1.96	8.55
40	127	9.65	68	149	61.1	1.35	2.99
45	142	11.2	66	146	74.7	1.19	1.44
50	160	12.2	61	597	98.9	1.61	6.38

Figure 18 shows the observed and inferred runtime distributions for selected sets from this experiment.

Task network complexity experiment We have examined variability due to the number of tasks, the number of bids, and the number of tasks in an auction. There are many other potential sources of variability, some of which can be measured or estimated prior to submitting an RFQ, and some of which clearly cannot (for example, the number of bids in an optimal solution). One parameter that is easy to measure is the density of precedence constraints in the

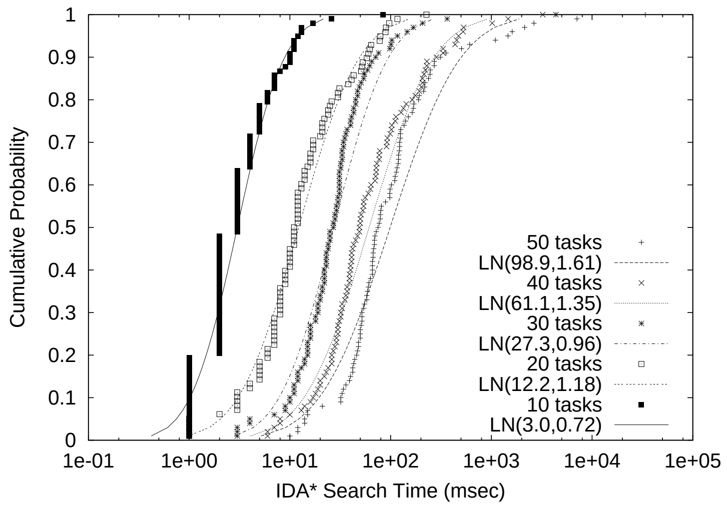

Fig. 18. IDA* run-time distributions for problems with 10 to 50 tasks, about 3.1 bids/task.

task network. However, given the way we generate plans and bids, it is not easy to build experiments that control this parameter independently.

In all experiments reported so far, the "Branch Factor" (see the definition in Section 4.1) has been set to a value of 2.4. The example task network shown in Figure 13 was built with this value. In this experiment, the branch factor is varied from 1.0 to 4.0 in increments of 1.0. Because root tasks have no predecessors, and because of the elimination of redundant precedence links, the actual number of precedence relationships per task is generally much lower than the branch factor. The results are shown in Table 6, where the first column, labeled "Fan In," gives the actual mean number of precedence links per task. There were 35 tasks in each set of 100 problems.

Table 6. Task network complexity experiment for the IDA* solver.

Fan In	Bid Count	Bid Size	# Solved (of 100)	Mean Time (ms)	σ	m	v	χ^2 (9 dof)
0.489	112	2.01	86	76100	516,000	505	6.74	0.966
0.876	111	5.06	100	427	2000	75.8	2.15	0.974
1.371	109	11.9	100	30.5	22.8	22.9	0.632	0.972
1.735	109	16.0	100	20.7	14.8	15.9	0.552	0.930
1.735	109	2.65	56	365,000	2,044,000	1975	9.85	1.000

As is evident from the table, the way bids are generated in our experimental setup causes bid count to go down as the number of precedence links is reduced. This is likely to cause a conflation of the network complexity factor with bid count effects. The probability distributions for this set are shown in Figure 19.

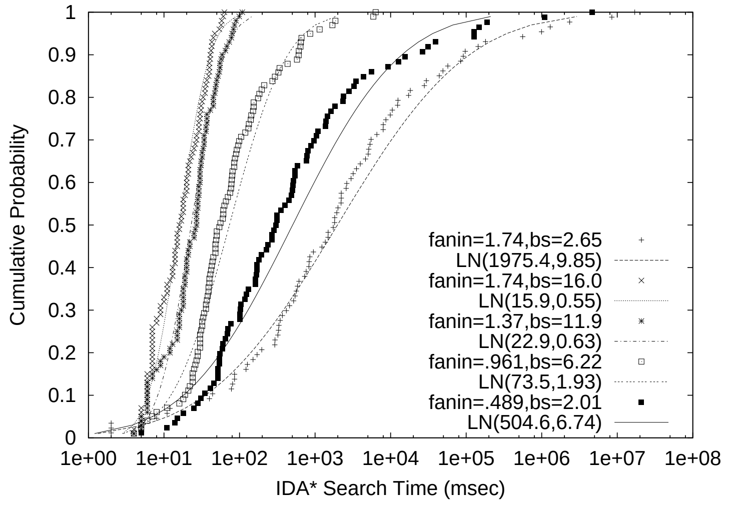

Fig. 19. Observed and inferred runtime distributions for the IDA* solver across a range of task network complexity values.

5 Related Work

This work draws from several fields. In Computer Science, it is related to work in artificial intelligence and autonomous agents. In Economics, it draws from auction theory and expected utility theory. From Operations Research, we draw from work in combinatorial optimization.

5.1 Multi-agent negotiation

MAGNET proposes using an auction paradigm to support problem-solving interactions among autonomous, self-interested, heterogeneous agents. Several other approaches to multi-agent problem-solving have been proposed. Some of them use a "market" abstraction, and some do not.

Rosenschein and Zlotkin [18] show how the behavior of agents can be influenced by the set of rules system designers choose for their 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, in a more or less "equitable" fashion, but not to have agents pay other agents for work. They also assume that each agent has sufficient resources to handle all the tasks, while we assume the contrary.

In Sandholm's TRACONET system [19, 20], agents redistribute work among themselves using 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. MAGNET agents are heterogeneous, and in general do not have the resources or capabilities to carry out the tasks necessary to meet their own goals without assistance from others.

Planning systems (see, for instance, [21–23]) assume multiple agents that operate independently. However, in those systems the agents are explicitly cooperative, and all work toward the achievement of a shared goal. MAGNET agents are trying to achieve their own goals and to maximize their own profits; there is no global or shared goal.

Solving problems using markets and auctions MAGNET uses an auction-based negotiation style because auctions have the right economic and motivational properties to support "reasonable" resource allocations among heterogeneous, self-interested agents. However, MAGNET uses the auction approach not only to allocate resources, but also to solve constrained scheduling problems.

A set of auction-based protocols for decentralized resource-allocation and scheduling problems is proposed in [24]. The analysis assumes that the items in the market are individual discrete time slots for a single resource, although there is a brief analysis of the use of the Generalized Vickrey Auctions [25] to allow for combinatorial bidding. A combinatorial auction mechanism for dynamic creation of supply chains was proposed and analyzed in [26]. This system deals with the constraints that are represented by a multi-level supply-chain graph, but does not deal with temporal and precedence constraints among tasks. MAGNET agents must deal with multiple resources and continuous time, but we do not currently deal explicitly with multi-level supply chains[6]

Several proposed bidding languages for combinatorial auctions allow bidders to express constraints, for example [27, 4]. However, these approaches only allow bidders to communicate constraints to the bid-taker (suppliers to the customer, in the MAGNET scenario), while MAGNET needs to communicate constraints in both directions.

Infrastructure support for negotiation Markets play an essential role in the economy [28], and market-based architectures are a popular choice for multiple agents (see, for instance, [29–32] and our own MAGMA architecture [33]). Most market architectures limit the interactions of agents to manual negotiations, direct agent-to-agent negotiation [20, 34], or some form of auction [35].

[6] Individual MAGNET agents can deal with multi-level supply chains by subcontracting, but this requires that the initial time allocation provide sufficient slack for the extra negotiation cycles.

The Michigan Internet AuctionBot [35] is a very interesting system, in that it is highly configurable, able to handle a wide variety of auction rules. It is the basis for the ongoing Trading Agent Competition [36], which has stimulated interesting research on bidding behavior in autonomous agents, such as [37].

Matchmaking, the process of making connections among agents that request services and agents that provide services, will be an important issue in a large community of MAGNET agents. The process is usually done using one or more intermediaries, called middle-agents [38]. Sycara et al. [39] present a language that can be used by agents to describe their capabilities and algorithms to use it for matching agents over the Web. Our system casts the Market in the role of matchmaker.

The MAGNET market infrastructure depends on an Ontology to describe services that can be traded and the terms of discourse among agents. There has been considerable attention to development of detailed ontologies for describing business and industrial domains (see, for instance, [40, 41]).

5.2 Combinatorial auctions

Determining the winners of a combinatorial auction [42] is an $\mathcal{NP}$-complete problem, equivalent to the weighted bin-packing problem. A good overview of the problem and approaches to solving it is [43]. Dynamic programming [44] works well for small sets of bids, but it does not scale well, and it imposes significant restrictions on the bids. Sandholm [12, 45] relaxes some of the restrictions and presents an algorithm for optimal selection of combinatorial bids, but his bids specify only a price and a set of items. Hoos and Boutilier[46] 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[7]. Andersson et al. [47] describe an Integer Programming approach to the winner determination problem in combinatorial auctions. Nisan [27] extends this model to handle richer bidding languages for combinatorial auctions, and we have extended it to handle the MAGNET situation in [9]. More recently, Sandholm [45] has described an improved winner-determination algorithm called BOB 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.

One of the problems with combinatorial auctions is that they are nearly always run in a single round sealed-bid format, and this is the format MAGNET uses. Parkes and Ungar [48] have shown how to organize multiple-round combinatorial auctions. Another problem is that the individual items in a combinatorial auction are individual items; there is no notion of quantity. MAGNET

[7] Under the "free disposal" assumption, the goal is to maximize revenue even if this means failing to allocate all the items at auction.

will eventually need to address this. This limitation is overcome in [49] for simple items without side constraints. The addition of precedence constraints would seriously complicate their procedure, and it has not yet been attempted.

5.3 Deliberation scheduling

The principal reason we are interested in search performance is because the search is embedded in a real-time negotiation scenario, and time must be allocated to it before bids are received, and therefore before the exact dimensions of the problem are known. In [50], deliberation scheduling is done with the aid of anytime and contract algorithms, and performance profiles. An anytime algorithm is one that produces a continuously-improving result given additional time, and a contract algorithm is one that produces a result of a given quality level in a given amount of time, but may not improve given additional time. The best winner-determination algorithms we know of for the MAGNET problem have marginal anytime characteristics, and we know of no applicable contract-type algorithms. In fact, [12] presents an inapproximability result for the winner-determination problem, leading us to believe that there may not be an acceptable contract algorithm.

One way to think about deliberation scheduling is to assign the time required for deliberation a cost, and then to balance the cost of deliberation against the expected benefit to be gained by the results of the deliberation. This is the approach taken in [51]. However, much of this analysis assumes that there is a “default” action or state that can be used or attained without spending the deliberation effort, and that there is a clear relationship between the time spent in deliberation and the quantifiable quality of the result. In the MAGNET case, the alternative to deliberation is to do nothing.

6 Conclusions

We have examined the problem of rational economic agents who must negotiate among themselves in a market environment in order to acquire the resources needed to accomplish their goals. We are interested in agents that are self-interested and heterogeneous, and we assume that a plan to achieve an agent’s goal may be described in the form of a *task network*, containing task descriptions, precedence relationships among tasks, and time limits for individual tasks. Negotiation among agents is carried out by holding combinatorial reverse auctions in a marketplace, in which a *customer* agent offers a task network in the form of a *request for quotes* (RFQ). *Supplier* agents may then place bids on portions of the task network, each bid specifying the tasks they are interested in undertaking, durations and time limits for those tasks, and a price for the bid as a whole. The presence of temporal and precedence constraints among the items at auction requires extensions to the standard winner-determination procedures for combinatorial auctions, and the use of the enhanced winner-determination procedure within the context of a real-time negotiation requires us to be able to predict its runtime when planning the negotiation process.

There are a number of real-world business scenarios where such a capability would be of value. These include flexible manufacturing, mass customization, travel arrangement, logistics and international shipping, health care resource management, and large-scale systems management. Each of these areas is characterized by limited capabilities and suboptimal performance, due at least in part to the limits imposed by human problem-solving capabilities. In each of these areas, a general ability to coordinate plans among multiple independent suppliers would be of benefit, but does not exist or is not used effectively because of an inability to solve the resulting combinatorial problems. The use of extended combinatorial auctions such as we propose is one approach to solving these problems. There are many difficulties yet to be overcome before this vision can be realized, however, not the least of which is that such auction-based markets would not be effective without wide adoption of new technology across an industry, and a willingness to delegate at least some level of autonomy and authority to that new technology.

We have designed and implemented a testbed, which we call MAGNET for Multi-AGent NEgotiation Testbed, to begin exploring and testing this vision. It includes a customer agent, a rudimentary market infrastructure, and a simple simulation of a population of supplier agents. The customer agent implementation is designed so that virtually all behaviors can be specified and implemented in terms of responses to events. Events can be external occurrences, internal state changes, or the arrival of a particular point in time. The MAGNET software package is available to the research community under an open-source license.

When a goal arises, the agent and its principal must develop a plan, in the form of a *task network*. Once a plan is available, a *bid-process plan* must be developed to guide the negotiation process. The bid-process plan specifies which tasks are to be offered in which markets, allocates time to the bidding process and to the plan execution, and may split the bidding into phases in order to mitigate risk. For each bidding step in the bid-process plan, time must be allocated to the customer to compose its RFQ, to the supplier to compose bids, and to the customer to evaluate bids. For each auction episode specified in the bid-process plan, a RFQ must be composed. The RFQ specifies a subset of tasks in the task network, and for each task, it specifies a *time window* within which that task must be accomplished. The setting of time windows is critical, because it influences the likelihood that bidders will bid, the prices bidders are likely to charge, and the difficulty of the resulting winner-determination process. If the time windows specified in the RFQ allow task precedence relationships to be violated, then the winner-determination process will need to choose a set of bids that can be composed into a feasible schedule. Once the RFQ has been issued and bids received, the agent must determine winners. We have described an optimal algorithm for determining winners based on an IDA* framework.

Because the winner-determination problem must be solved within a predetermined period of time, it is important to have a clear idea of how much time to allocate to it, and to know what parameters to use in predicting its run time. We therefore conducted a series of experiments to characterize the performance of our winner-determination method over a variety of problem sizes and shapes. The goal was to determine probability distributions, so that we could allocate the time necessary to achieve a known probability of solving the problem. We found that the lognormal distribution was a good model of search performance.

Much work remains to be done before the vision of the MAGNET project is fully realized. Some of that work, particularly with respect to the supplier agent and its decision processes, is already under way by other members of the team.

With respect to the customer agent, many of the decision processes outlined in Section 2 still need to be worked out and tested. The present work has resulted in models for the auction winner-determination problem and the time that must be allocated to it. For the remainder of the decisions, we need models that will maximize the expected utility of the agent or its principal. These include composing the plan, developing the bid-process plan, allocating time to the deliberation processes of the customer and suppliers, balancing negotiation time against plan execution time, setting the time windows in the RFQ, scheduling the work in preparation for awarding bids, and dealing with unexpected events during plan execution. Babanov et al. [5, 52] have addressed the problem of setting time windows in the customer's RFQ.

The language we currently use for plans and bids treats tasks as simple atomic objects, without attributes. There are many real-world problems in which attributes are important, both for specifying tasks and for expressing offers in bids. Examples include colors, quantities, dimensions, and quality attributes. In addition, many real-world operations operate on a "flow" basis. This includes the wine-making example we used in Chapter 2, in which the precedence between filling bottles and applying labels would normally be applied bottle-by-bottle, and not at the batch level. In addition, the expressivity of our bidding language is limited. A number of proposals have been made for more expressive bidding languages in combinatorial auctions [27, 4]. Bidding can also be done with *oracles*, which are functions passed from bidder to customer that can be evaluated to produce bid conditions. Some features of a more expressive bidding language would likely have minimal impact on the winner-determination process (parameterized quality values, for example), while others, including the use of oracles, could require wholesale re-invention.

References

1. Collins, J., Tsvetovat, M., Mobasher, B., Gini, M.: MAGNET: A multi-agent contracting system for plan execution. In: Proc. of SIGMAN, AAAI Press (1998) 63–68
2. Collins, J., Bilot, C., Gini, M., Mobasher, B.: Mixed-initiative decision support in agent-based automated contracting. In: Proc. of the Fourth Int'l Conf. on Autonomous Agents. (2000) 247–254
3. Nisan, N.: Bidding and allocation in combinatorial auctions. Technical report, Institute of Computer Science, Hebrew University (2000)

4. Boutilier, C., Hoos, H.H.: Bidding languages for combinatorial auctions. In: Proc. of the 17th Joint Conf. on Artificial Intelligence. (2001) 1211–1217
5. Babanov, A., Collins, J., Gini, M.: Asking the right question: Risk and expectation in multi-agent contracting. Artificial Intelligence for Engineering Design, Analysis and Manufacturing **17** (2003) 173–186
6. Collins, J., Jamison, S., Gini, M., Mobasher, B.: Temporal strategies in a multi-agent contracting protocol. In: AAAI-97 Workshop on AI in Electronic Commerce. (1997)
7. Hillier, F.S., Lieberman, G.J.: Introduction to Operations Research. McGraw-Hill (1990)
8. Parkes, D.C., Ungar, L.H.: An auction-based method for decentralized train scheduling. In: Proc. of the Fifth Int'l Conf. on Autonomous Agents, Montreal, Quebec, ACM Press (2001) 43–50
9. Collins, J., Gini, M.: An integer programming formulation of the bid evaluation problem for coordinated tasks. In Dietrich, B., Vohra, R.V., eds.: Mathematics of the Internet: E-Auction and Markets. Volume 127 of IMA Volumes in Mathematics and its Applications. Springer-Verlag, New York (2001) 59–74
10. Collins, J., Bilot, C., Gini, M., Mobasher, B.: Decision processes in agent-based automated contracting. IEEE Internet Computing **5** (2001) 61–72
11. Sandholm, T.: An algorithm for winner determination in combinatorial auctions. In: Proc. of the 16th Joint Conf. on Artificial Intelligence. (1999) 524–547
12. Sandholm, T.: Algorithm for optimal winner determination in combinatorial auctions. Artificial Intelligence **135** (2002) 1–54
13. Korf, R.E.: Depth-first iterative deepening: An optimal admissible tree search. Artificial Intelligence **27** (1985) 97–109
14. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Upper Saddle River, NJ (1995)
15. Collins, J., Demir, G., Gini, M.: Bidtree ordering in IDA* combinatorial auction winner-determination with side constraints. In Padget, J., Shehory, O., Parkes, D., Sadeh, N., Walsh, W., eds.: Agent Mediated Electronic Commerce IV. Volume LNAI2531., Springer-Verlag (2002) 17–33
16. Law, A.M., Kelton, W.D.: Simulation Modeling and Analysis. Second edn. McGraw-Hill (1991)
17. Sandholm, T., Suri, S., Gilpin, A., Levine, D.: CABOB: A fast optimal algorithm for combinatorial auctions. In: Proc. of the 17th Joint Conf. on Artificial Intelligence, Seattle, WA, USA (2001) 1102–1108
18. Rosenschein, J.S., Zlotkin, G.: Rules of Encounter. MIT Press, Cambridge, MA (1994)
19. Sandholm, T.W., Lesser, V.: On automated contracting in multi-enterprise manufacturing. In: Distributed Enterprise: Advanced Systems and Tools, Edinburgh, Scotland (1995) 33–42
20. Sandholm, T.W.: Negotiation Among Self-Interested Computationally Limited Agents. PhD thesis, Department of Computer Science, University of Massachusetts at Amherst (1996)
21. Pollack, M.E.: Planning in dynamic environments: The DIPART system. In Tate, A., ed.: Advanced Planning Technology. AAAI Press (1996)
22. Wilkins, D.E., Myers, K.L.: A multiagent planning architecture. In: Proc. Int'l Conf. on AI Planning Systems. (1998) 154–162
23. Cox, J.S., Durfee, E.H., Bartold, T.: A distributed framework for solving the multiagent plan coordination problem. In: Autonomous Agents and Multi-Agent Systems. (2005) 821–827

24. Wellman, M.P., Walsh, W.E., Wurman, P.R., MacKie-Mason, J.K.: Auction protocols for decentralized scheduling. Games and Economic Behavior **35** (2001) 271–303
25. Varian, H.R., MacKie-Mason, J.K.: Generalized vickrey auctions. Technical report, University of Michigan (1995)
26. Walsh, W.E., Wellman, M., Ygge, F.: Combinatorial auctions for supply chain formation. In: Proc. of ACM Conf on Electronic Commerce (EC'00). (2000) 260–269
27. Nisan, N.: Bidding and allocation in combinatorial auctions. In: Proc. of ACM Conf on Electronic Commerce (EC'00), Minneapolis, Minnesota, ACM SIGecom, ACM Press (2000) 1–12
28. Bakos, Y.: The emerging role of electronic marketplaces on the Internet. Comm. of the ACM **41** (1998) 33–42
29. Chavez, A., Maes, P.: Kasbah: An agent marketplace for buying and selling goods. In: Proc. of the First Int'l Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK, Practical Application Company (1996) 75–90
30. Rodriguez, J.A., Noriega, P., Sierra, C., Padget, J.: FM96.5 - A Java-based electronic auction house. In: Second Int'l Conf on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97), London (1997) 207–224
31. Sycara, K., Pannu, A.S.: The RETSINA multiagent system: towards integrating planning, execution, and information gathering. In: Proc. of the Second Int'l Conf. on Autonomous Agents. (1998) 350–351
32. Wellman, M.P., Wurman, P.R.: Market-aware agents for a multiagent world. Robotics and Autonomous Systems **24** (1998) 115–125
33. Tsvetovatyy, M., Gini, M., Mobasher, B., Wieckowski, Z.: MAGMA: An agent-based virtual market for electronic commerce. Journal of Applied Artificial Intelligence **11** (1997) 501–524
34. Faratin, P., Sierra, C., Jennings, N.R.: Negotiation decision functions for autonomous agents. Int. Journal of Robotics and Autonomous Systems **24** (1997) 159–182
35. Wurman, P.R., Wellman, M.P., Walsh, W.E.: The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In: Second Int'l Conf. on Autonomous Agents. (1998) 301–308
36. Collins, J., Arunachalam, R., Sadeh, N., Ericsson, J., Finne, N., Janson, S.: The supply chain management game for the 2005 trading agent competition. Technical Report CMU-ISRI-04-139, Carnegie Mellon University, Pittsburgh, PA 15213 (2004)
37. Stone, P., Schnapire, R.E., Csirik, M.L.L.J.A., McAllester, D.: ATTac-2001: A learning, autonomous bidding agent. Submitted to the Eigtheenth National Conference on Artificial Intelligence (AAAI-2002) (2002)
38. Sycara, K., Decker, K., Williamson, M.: Middle-agents for the Internet. In: Proc. of the 15th Joint Conf. on Artificial Intelligence. (1997) 578–583
39. Sycara, K., Klusch, M., Widoff, S., Lu, J.: Dynamic service matchmaking among agents in open information environments. SIGMOD Record (ACM Special Interests Group on Management of Data) **28** (1999) 47–53
40. Fox, M., Barbuceanu, M., Teigen, R.: Agent-oriented supply-chain management. The International Journal of Flexible Manufacturing Systems **12** (2000) 165–188
41. Chandra, C., Tumanyan, A.: Supply chain system taxonomy: development and application. In: Proc. 12th Annual Industrial Engineering Research Conference IERC-2003, Portland, Oregon, USA (2003)

42. McAfee, R., McMillan, P.J.: Auctions and bidding. Journal of Economic Literature **25** (1987) 699–738
43. de Vries, S., Vohra, R.: Combinatorial auctions: a survey. Technical report, Technische Universität München (2001)
44. Rothkopf, M.H., Pekeč, A., Harstad, R.M.: Computationally manageable combinatorial auctions. Management Science **44** (1998) 1131–1147
45. Sandholm, T., Suri, S.: Bob: Improved winner determination in combinatorial auctions and generalizations. Artificial Intelligence **145** (2003) 33–58
46. Hoos, H.H., Boutilier, C.: Solving combinatorial auctions using stochastic local search. In: Proc. of the Seventeen Nat'l Conf. on Artificial Intelligence. (2000) 22–29
47. Andersson, A., Tenhunen, M., Ygge, F.: Integer programming for combinatorial auction winner determination. In: Proc. of 4th Int'l Conf on Multi-Agent Systems. (2000) 39–46
48. Parkes, D.C., Ungar, L.H.: Iterative combinatorial auctions: Theory and practice. In: Proc. of the Seventeen Nat'l Conf. on Artificial Intelligence. (2000) 74–81
49. Leyton-Brown, K., Shoham, Y., Tennenholtz, M.: An algorithm for multi-unit combinatorial auctions. In: Proc. of the Seventeen Nat'l Conf. on Artificial Intelligence, Austin, Texas (2000)
50. Greenwald, L., Dean, T.: Anticipating computational demands when solving time-critical decision-making problems. In Goldberg, K., Halperin, D., Latombe, J., Wilson, R., eds.: The Algorithmic Foundations of Robotics. A. K. Peters, Boston, MA (1995)
51. Boddy, M., Dean, T.: Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. Artificial Intelligence **67** (1994) 245–286
52. Babanov, A., Collins, J., Gini, M.: Harnessing the search for rational bid schedules with stochastic search and domain-specific heuristics. In: Proc. of the Third Int'l Conf. on Autonomous Agents and Multi-Agent Systems, New York, AAAI Press (2004) 269–276