Multi-Agent Contracting for Supply-Chain Management

John Collins*, Rashmi Sundareswara, Maksim Tsvetovat†,
Maria Gini, and Bamshad Mobasher‡

Department of Computer Science and Engineering
University of Minnesota

Abstract

We present a system for multi-agent contract negotiation, implemented as a generalized market architecture called MAGNET. MAGNET provides support for a variety of types of transactions, from simple buying and selling of goods and services to complex multi-agent contract negotiations. In the latter case, MAGNET is designed to negotiate contracts based on temporal and precedence constraints, and includes facilities for dealing with time-based contingencies. The market operates as an explicit intermediary in the negotiation process, which helps in controlling fraud and discouraging counterspeculation.

We introduce a multi-criterion, anytime bid evaluation strategy that incorporates cost, task coverage, temporal feasibility, and risk estimation into a simulated annealing framework. We report on an experimental evaluation using a set of increasingly informed search heuristics within simulated annealing. The results show that excess focus on improvement leads to faster improvement early on, at the cost of a lower likelihood of finding a solution that satisfies all the constraints.

1 Introduction

Over the past decade, the complexity of logistics involved in manufacturing has been increasing nearly exponentially. Many processes are being outsourced to outside contractors, making supply chains longer and more convoluted. The increased complexity is often compounded by accelerated production schedules which demand tight integration of all processes. Thus, the field is ripe for the introduction of systems that automate logistics planning among multiple entities such as manufacturers, part suppliers and specialized subcontractors.

To help automate logistics planning, we propose a testbed for multi-agent contract negotiation, implemented as a generalized market architecture called MAGNET (Multi AGent

*jcollins@cs.umn.edu
†Currently at the Robotics Institute, Carnegie Mellon University
‡Currently at DePaul University

NEgotiation Testbed). MAGNET provides support for a variety of types of transactions, from simple buying and selling of goods and services to complex multi-agent contract negotiations. In the latter case, MAGNET is designed to negotiate contracts based on temporal and precedence constraints, and includes facilities for dealing with time-based contingencies.

MAGNET adds a new dimension to business-to-business interactions, by adding the ability to automate the negotiation and execution of complex combinations of contracts among multiple suppliers. This is especially important for the coordination of supply-chain management with production scheduling.

Companies usually work with prequalified suppliers and do not rely on auctions to get the commodities or services they need. Buyer-supplier relationships depend on factors such as quality, delivery performance, and flexibility as opposed to just cost [19], and these must be taken into account if automated negotiation is to be successful. Current Internet-based procurement systems are mostly limited to non production related procurement, such as office supplies or computer equipment [14]. With most organizations spending at least one third of their budget to purchase goods or services, procurement savings have a significant business value. Business-to-business transactions over the Internet are expected to increase to $1.3 trillion in 3 years, according to Forrester Research, much more than the $108 billion expected from consumer online spending.

One of the more difficult problems an agent faces in dealing with negotiation over complex plans is the problem of evaluating bids. The agent must solve both bid-allocation and temporal feasibility constraints, while attempting to minimize cost and risk. We have developed a highly tunable anytime search, based on a Simulated Annealing [29] framework, using a set of modular selectors and evaluators. Given that the time allocated to search will seldom be sufficient to explore a significant fraction of the search space, we must find an appropriate tradeoff between systematic optimization and random exploratory behavior. We describe here the first of a set of experiments that will allow us to allocate time to the various agent activities, and to maximize the performance of the search.

This paper is organized as follows: in Section 2 we identify the requirements for a generalized multi-agent market architecture that can support complex agent interactions, and in Section 3 we present a novel architecture that satisfies these requirements. Section 4 describes the decision processes faced by a customer agent in the MAGNET environment. Section 5 presents the results of a set of experiments that attempt to characterize the bid evaluation search process. Section 6 describes the state of the current implementation. Section 7 relates our work to other published research. Finally, in Section 8, we conclude and discuss further research opportunities.

2 The Role of an Independent Market

The MAGNET architecture, described in the next section, provides a framework for secure and reliable commerce among self-interested agents. There are several types of services that are difficult for self-interested agents to provide for themselves, and that have not been fully addressed in existing systems. These include negotiation support, value-added services, and security issues.

Support for multi-agent negotiation over extended time periods. Negotiations may require extended periods of time to complete, during which a context must be maintained. The time during which the negotiated transaction extends can also span significant periods of time, in the range of weeks to months. In order to support this, the market infrastructure maintains the state of each transaction over time.

This is a prerequisite for many other functions the market will perform, and it makes the system more robust in the face of hardware and communication failures. Most negotiation protocols involve time limits, such as a deadline for receipt of bids. All parties to a time-sensitive negotiation process must have a common time reference. The market architecture provides this. The architecture also has the ability to validate certain types of non-performance, and to assess negotiated penalties.

Value-added services. An independent market entity can also provide a number of value-added services that may be commonly used by suppliers and customers. For example, the market can provide matchmaking facilities to bring together suppliers and customers based on their stated preferences [40, 41]. This is particularly important for two reason. First, participants can continuously issue or retract registered capabilities in various domains. This makes it difficult for individual participating agents to keep track of and have access to the most up-to-date information. Secondly, such a facility provides a form of filtering and reduce the computational costs during bid evaluation by customer agents.

Furthermore, the market may serve as a repository of statistical data about various participants. This may include general statistics about availability of suppliers with specific capabilities, or independent ratings for both suppliers and customers based on past performance. The general statistical information is useful for customer agents in formulating a request-for-quotes, while performance ratings can be used by both customer and supplier agents to determine the risk (or price) associated with various bids. Finally, the market can provide publish-subscribe facilities to provide registered participants with notification of important events.

Protection against fraud and misrepresentation. We must assume that participating agents will take advantage of any opportunities that exist in the design of the market to gain advantage. The structure of the market recognizes and protects against situations that allow agents to gain unfair advantage at the expense of other agents. Strategies that can result in this type of "unfair" gain include:

- Hiding one's identity or taking on the identity of another. This includes changing identity in order to escape the consequences of poor service on prior commitments.
- Dishonest auctioneer - In Vickrey-type auctions [46, 45], the motivation for truth-telling on the part of participants is predicated on their belief in the honesty of the auctioneer.
- Miscommunication of the rules under which a transaction is being conducted.
- Failure to follow through on commitments.

Discouragement of counterspeculation. Opportunities for counterspeculation arise when the rules of negotiation allow agents to gain advantage by making use of factors other than their own capabilities and valuations, such as their estimates of the capabilities and valuations of the customers or other suppliers [21]. We are concerned with two general types of counterspeculation. Value-based counterspeculation [32, 35, 46, 45] occurs when agents use their own estimates of each other's valuations to set bid prices. In [7], we identified two classes of time-based counterspeculation opportunities in a contracting domain that can be controlled by the settings of certain timing parameters. One of these situations occurs when supplier agents are allowed to expire their bids before the customer's request for quotes expires. This forces customers to make decisions without full information on other, possibly more advantageous bids. The other situation occurs when suppliers believe that the customer will start the bid evaluation process before all bids are received. If the supplier believes that the customer's resource limitations will prevent full consideration of all bids, then early submission of bids, at potentially higher prices, can be used to skew the customer's reasoning process.

MAGNET provides a neutral third-party facility for controlling and filtering protocol exchanges that can reduce or prevent both value-based and temporal counterspeculation. It is up to the agents to decide the extent to which these facilities are used, since they may slow the negotiation process or reduce the information exchange.

3 The MAGNET Architecture

The MAGNET system is based on these elements:

1. The *market.* Agents find each other and carry out negotiations through a distributed infrastructure that enforces protocol rules, limits opportunities for fraud and counterspeculation, provides a common information base for agents, and tracks the requests, commitments, and progress toward goals among the agent population.

2. The *agents.* Each agent is an independent entity, with its own goals and resources. In general, the resources under control of an individual agent are not sufficient to satisfy its goals, and so the agent must negotiate with other agents.

Because it is based on a market-based economic model, a MAGNET system acts to allocate resources among a community of agents to their highest-value uses, over time, and in a completely distributed fashion. Because MAGNET agents are heterogeneous and self-interested, they may represent real-world entities who may tune their levels of cooperation and competitiveness to suit their own needs.

3.1 The MAGNET Market Infrastructure

In order to provide these capabilities, the MAGNET system incorporates a distributed, hierarchical market infrastructure that manages security, tracks commitments and performance of agents, and logs their interactions. The market specifies the terms of discourse among

agents through an *Ontology*, and keeps historical data that may be used by agents in their risk evaluations. Different market segments may specify specialized ontologies. Agents who wish to offer resources and services do so through one or more market segments whose ontologies describe their offerings.

The fundamental elements of the market architecture are the *exchange*, the *market*, and the *market session*.

3.1.1 The Exchange

An *exchange* is a collection of domain-specific markets in which goods and services are traded, along with some generic services required by all markets, such as verifying identities of participants in a transaction, and a Better Business Bureau that can provide information about the reliability of other agents based on past performance. Architecturally, an exchange is a network-accessible resource that supports a set of markets and common services, as depicted in Figure 1.

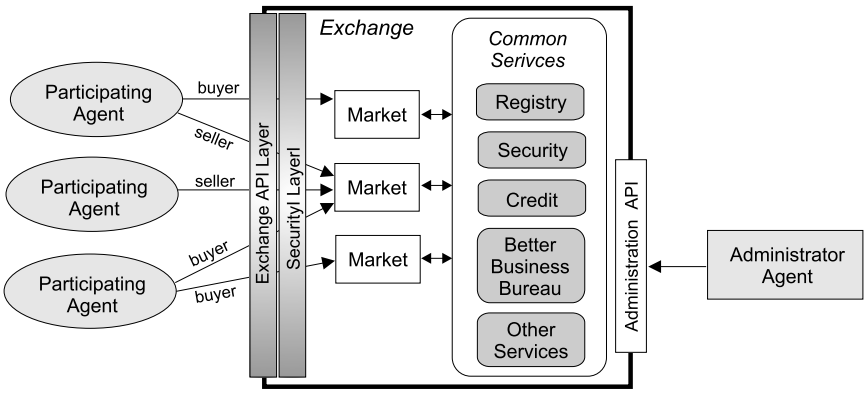

Figure 1: The Structure of an Exchange

3.1.2 Markets

Each *market* within an exchange is a forum for commerce in a particular commodity or business area. There would be markets devoted to banking, publishing and printing, construction, transportation, industrial equipment, etc. Each market includes a set of domain-specific services and facilities, as shown in Figure 2, and each market draws upon the common services of the exchange.

An important component of each market is a set of current *market sessions* in which the actual agent interactions occur. Agents participating in a market may do so as either session initiators, or as clients, or both. As detailed in the next section, each session is initiated by a single agent for a particular purpose, and in general multiple agents may join an existing session as clients. Important elements of the market include:

- An *Ontology* that is specific to the domain of the market, specifying the terms of discourse within that domain. In a commodity-oriented domain, it would include terms

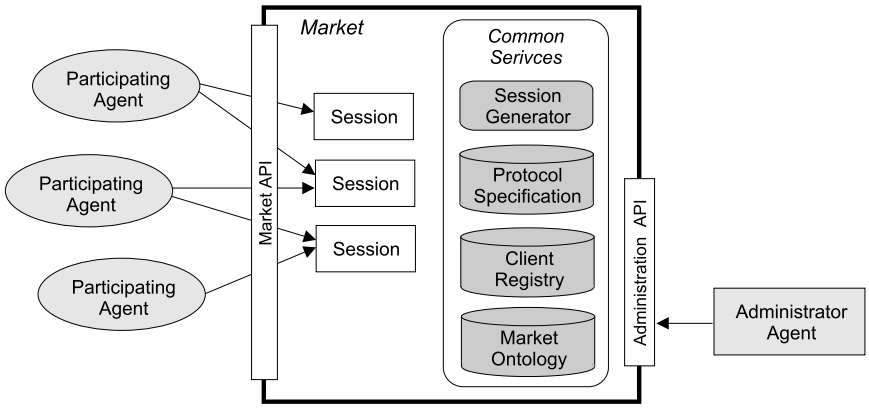

Figure 2: The Structure of a Market within the Exchange

for the products or services within the domain, as well as terminology for quality, quantity, features, terms and conditions of business, etc. In a planning-oriented domain, specifications of services would be in a form that supports planning, including, for example, preconditions, postconditions, and decomposition information.

- A *Protocol Specification* that formalizes the types of negotiation supported within the market. These are limits on parameters of the negotiation protocol, such as maximum decommitment, whether bids can be awarded before the bid deadline, etc.

- A *Registry* of market participants who have expressed interest in doing business in the market. Entries in this registry include the identity of a participant, a catalog (or a method for accessing a catalog) of that participant's interests, products or capabilities, which can be used for matchmaking [40, 41]. Catalogs are required to express interests and offerings in terms of the market's ontology.

3.1.3 Market Sessions

A *market session* (or simply a session) is the vehicle through which market services are delivered dynamically to participating agents. It serves as an encapsulation for a transaction in the market, as well as a persistent repository for the current state of the transaction.

We have chosen the term "session" to emphasize the temporally extended nature of many of these interactions. For example, in a contracting market, if an agent wishes to build a new house, it initiates a session and issues a Request for Quotes (RFQ). The session extends from the initial RFQ through the negotiation, awards, construction work, paying of bills, and final closing. In other words, the session encloses the full life of a contract or a set of related contracts. The session mechanism ensures continuity of partially-completed transactions, and relieves the participating agents from having to keep track of detailed negotiation status themselves.

Agents can play two different roles with respect to any session. The agent who initiates a session is known as the *session initiator*, while other participating agents are known as *session*

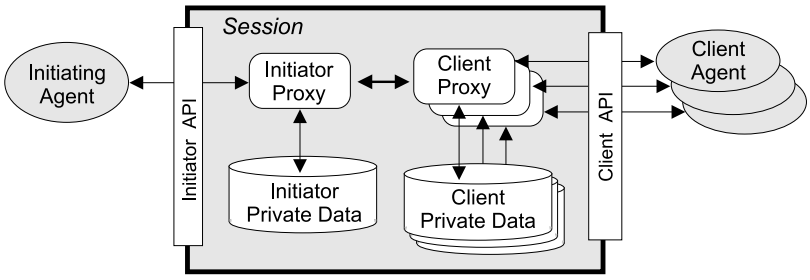

Figure 3: The Structure of a Market Session

clients. A session can be initiated either for the purpose of buying or selling, depending on the type of market. A session could also be initiated to sell items or services at auction.

At any given time, a session can be *open* to new participants, or *closed*. A public auction would typically be open to new participants, but most sessions are closed once the contracts are let. The market maintains a list of open sessions which may be accessed by agents.

Figure 3 shows the structure of a session. Two APIs are exposed, one for the session initiator and one for session clients. Each session contains an Initiator Proxy that implements the Initiator API and persistently stores the current state of the session from the standpoint of the initiator. A Client Proxy is provided for each client that similarly provides a Client API to the client agent, and persistently stores the current state of the session from the standpoint of the client. Proxies are market entities that act on behalf of the agents and enforce market rules.

There are two reasons for the existence of the proxy components. The first is related to security: client proxy components cannot see the private data of the initiator or of other clients. The second is that in a distributed system environment, the processing and persistent data elements of the initiator and clients would presumably be at different locations in the network to maximize performance. In Section 6 these proxy components are called *participants*.

3.2 The MAGNET Agents

In general, a MAGNET agent has three basic functions: planning, negotiation, and execution monitoring. Within the scope of a negotiation, we distinguish between two agent *roles*, the *Customer* and the *Supplier*. A Customer is an agent who has a goal to satisfy, and needs resources outside its direct control in order to achieve its goal. The goal may have a *value* that varies over time. A Supplier is an agent who has resources and who, in response to a RFQ, may offer to provide resources or services, for specified prices, over specified time periods. Figure 4 shows the general relationships among Customer agents, Supplier agents, and the Market.

The negotiation process consists of a *contracting* phase and a *execution* phase. The contracting phase is a simple three-step process consisting of a Request For Quotes, a Bidding cycle, and an Award cycle. We have developed a finite, leveled commitment protocol that limits the time and bandwidth required for the negotiation process without limiting the

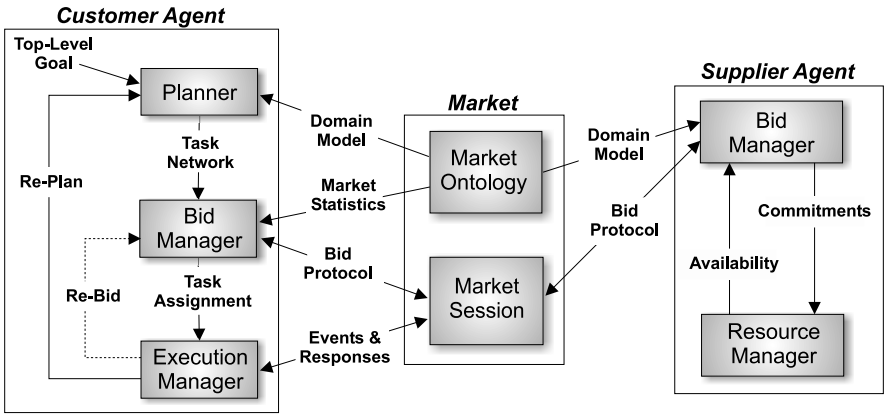

Figure 4: MAGNET Agents

scope of possible agreements [36].

The execution phase may involve negotiations over schedule adjustments, decommitments, and in some cases repeating the bidding cycle when it becomes necessary to re-allocate resources that had originally been committed.

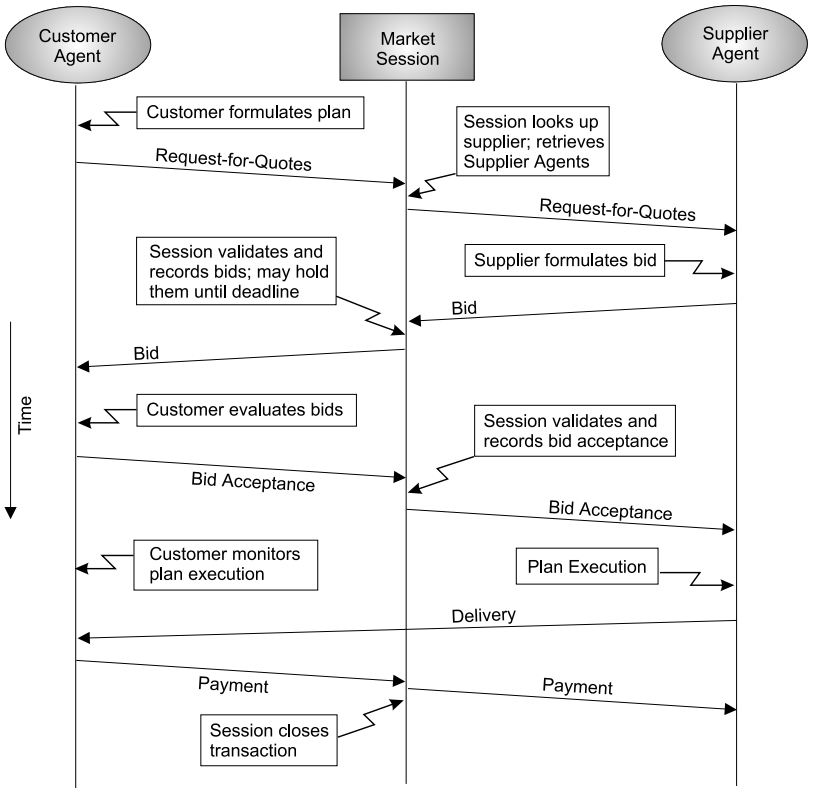

Figure 5: A Typical Session-Mediated Negotiation

The interactions involved in the basic bidding and execution cycle among the Customer, Supplier, and market session are illustrated in Figure 5. During execution, the interactions

can be much more complex than indicated here, since either party may decommit from a contract (and pay a penalty), and the Customer is continuously monitoring and repairing its plan by replanning and rebidding when events fail to proceed according to expectations.

Each bid may specify one or more tasks, including prices and time constraints for the individual tasks, as well as a *discount* or a *premium* for acceptance of the whole bid. Combination bids (bids that include multiple tasks) are interpreted as an exclusive-OR, so when multiple combination bids are submitted by the same supplier agent, only one of them can be accepted.

The Customer initiates the process by starting a market Session and issuing a RFQ. Once the Customer agent receives bids, it must evaluate them based on cost, task coverage, and time constraints, and select the optimal set of bids (or parts thereof) which can satisfy its goals. The resulting *task assignment* forms the basis of an initial schedule for the execution of the tasks. The Customer's goals are time-sensitive, the negotiation process requires Customer and Suppliers to agree on the times for execution of tasks, and time factors can affect the cost of execution.

The timeline in Figure 6 shows an abstract view of the progress of a single negotiation. At the beginning of the process, the Customer agent must allocate time to deliberation for its own planning, for supplier bid preparation, and for its own bid evaluation. 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. On the other hand, the Customer's ability to find a good set of bids is dependent on the time allocated to bid evaluation. These time intervals can be overlapped to some extent, but doing so creates opportunities for strategic manipulation of the Customer by the Suppliers, as we discussed earlier.

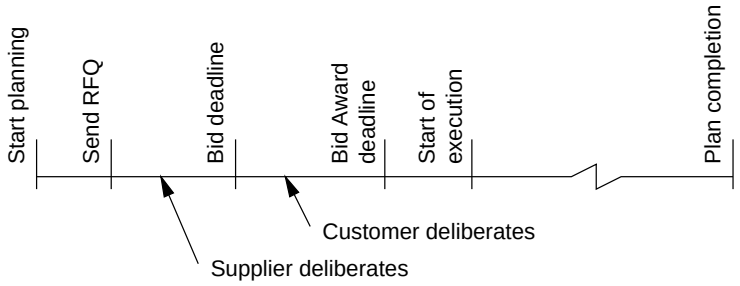

Figure 6: Agent Interaction Timeline

After a set of bids is selected, Suppliers are notified of their commitments, and the Customer agent invokes its Execution Manager to oversee completion of the plan. Plan maintenance behaviors can include monitoring supplier performance, re-negotiating existing commitments, re-bidding portions of the plan, re-planning for subgoals that are in jeopardy, and abandoning the goal.

4 Components and Activities of the Customer Agent

We now focus on the structure and responsibilities of a Customer agent in the MAGNET environment. The basic operations are planning, bidding, and plan execution.

4.1 Planner

The Planner's task is to turn high-level goals into executable plans, represented as task networks. A task network consists of a set of task descriptions of nonzero length, the temporal constraints among them, and possibly nonzero delays between tasks, to cover communication and transportation delays. An example task network is shown in Figure 7. The operations in the task network do not need to be linearized with respect to time, since operations can be executed in parallel by multiple agents. The task network is passed to the Bid Manager.

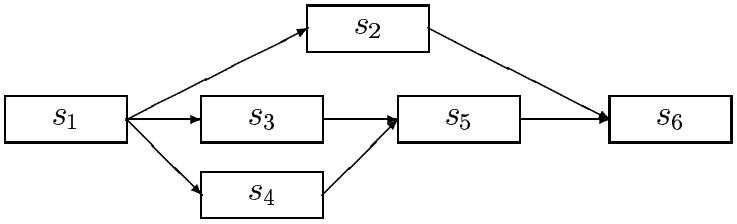

Figure 7: Example task network

4.2 Bid Manager

The Bid Manager is responsible for ensuring that resources are assigned to each of the tasks of a plan, that the assignments taken together form a feasible schedule, and that the cost and risk of executing the plan is minimized. This cost must also be less than the value of the goal at the time the goal is reached. When the Bid Manager is invoked, some tasks in the plan may already be assigned. This can occur because the Execution Manager may use the Bid Manager to repair a partially-completed plan in which previously determined assignments have failed, or because the agent will perform some of the tasks itself.

The Bid Manager must construct and issue a RFQ, evaluate bids, and accept bids in order to carry out its responsibilities.

4.2.1 Construct and Issue Request For Quotes

The RFQ contains a subset $\mathcal{S}$ of the tasks in the plan $\mathcal{P}$, with their precedence relations. As pointed out earlier, there might be elements of $\mathcal{P}$ that are not included in the RFQ. For each task $s \in \mathcal{S}$ in the RFQ the Bid Manager must specify:

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

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

We assume the agent has general knowledge of normal durations of tasks. One of the roles of the MAGNET market infrastructure is to gather and publish this information. In order to minimize bid prices while minimizing the overall duration of the plan, the Customer agent schedules tasks ahead of time using expected durations, computing early start and late finish times using the Critical Path (CPM) algorithm [20].

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

The tradeoff between minimizing plan duration and attracting usable bids from suppliers affects how slack should be set. Our method based on CPM is appropriate if adequate approximations of task durations are known, because it will reduce the likelihood of bids being rejected for failure to mesh with the overall schedule, and it will reduce the average bid prices to the extent that it reduces speculative resource commitments on the part of suppliers.

In Figure 8 the medium-gray bars show the expected durations of the tasks shown in Figure 7. The overall slack in the schedule is 5 units for a 35 unit makespan, or about 14%. We can reduce the task durations used to generate the time windows in the RFQ (lighter bars), leading to larger time windows and presumably greater flexibility for suppliers. This can work if the shortened durations used to generate this schedule have a reasonable statistical likelihood of occurring among the received bids. In this example, the lighter bars show use of durations that are 1 standard deviation below the expected values, while the black bars show an alternative formulation in which the total expected slack is apportioned among the tasks on the critical path to produce an RFQ that allows no overlap among adjacent tasks. The former is likely to produce more bids, and potentially lower-cost bids, because of the additional scheduling flexibility offered to suppliers. The latter will reduce the effort the customer must expend to compose a feasible plan, assuming bids are received to cover all the tasks.

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

4.2.2 Evaluate Bids

When bids are returned, the Bid Manager must assemble them into a minimum-cost feasible schedule in order to determine which bids to accept.

For each bid, the Bid Manager has the option of selecting the entire bid and paying an overall discounted price (or possibly a premium), or selecting a subset of the individual task

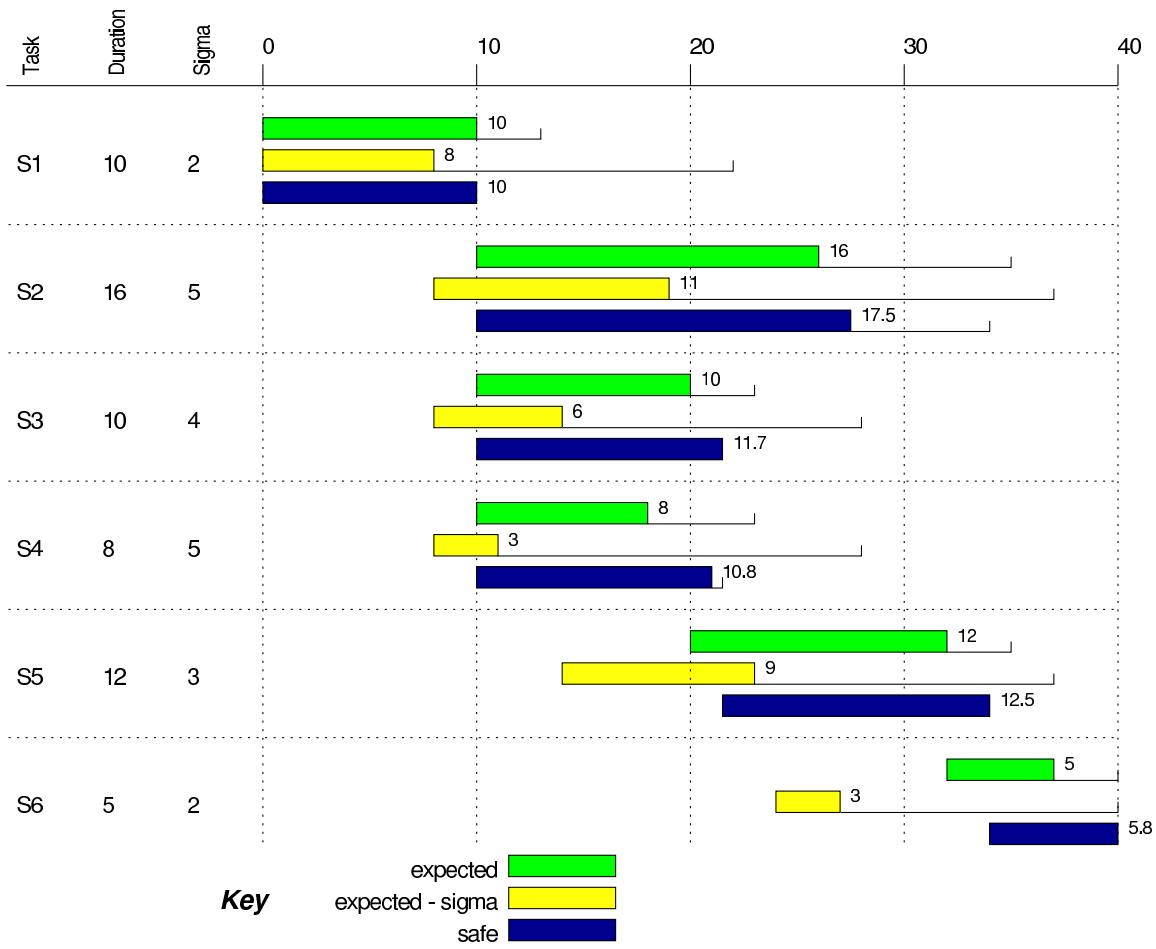

Figure 8: Reducing task durations to increase time windows

bids from a combination bid. Timing information for a bid includes early start, late finish, and duration for each task in the bid. Bids that cover multiple tasks are required to specify prices for each of the individual tasks, as well as a (possibly discounted) price for the entire set of tasks. The semantics of a bid is that a supplier is willing to perform the task or combination of tasks for the bid price, starting at any time in the time window specified in the bid.

The complexity of the bid evaluation algorithm depends on the type of bids allowed (single item, bundles, multi-attribute, etc). We allow agents to bid on combinations of items, we do not assume superadditivity of bid prices (the bid price for a combination of items is not necessarily higher or lower than the bid price for the individual items), and we use exclusive OR bids (when multiple bids on combinations are submitted by the same agent, only one of them can be accepted). In addition, bids can have multiple attributes and include time windows. This makes bid evaluation much more complex than in traditional auctions [25, 49, 34].

The time-dependent nature of the negotiation protocol requires that the search be completed within a fixed period of time. Boddy and Dean [2] have characterized this type of

Inputs:
$\mathcal{S}$: the set of tasks, with precedence relations, to be assigned
$\mathcal{B}$: the set of bids, including both composite and component bids
Output:
N_{opt}: the node having a mapping $N_{opt}.\mathcal{M}$ of bids to tasks
with the best known evaluation
Process:

```
for each s ∈ S /* pre-process bid-task mappings */
    compute the set of bids B_s ⊂ B such that ∀b ∈ B_s, b includes s
    compute s.p̄ /* the mean price for s among B_s */
if ∃s ∈ S such that B_s = ∅ then
    exit /* there is at least one task for which no bid was submitted */
Q ← priority queue of maximum length beam_width,
      sorted by node evaluation N.v
N_0 ← empty node with N_0.M = ∅
for each s ∈ S /* check for singletons */
    if length(B_s) = 1 then
        N_0.M ← N_0.M ∪ m(s, first(B_s))
              /* the function m(s,b) creates a mapping of bid b to task s */
N_0.v ← evaluate_node(N_0) /* see narrative in this section */
N_opt ← N_0 /* initialize the result */
insert(Q, N_0) /* insert the first node into the queue */
T ← T_0 /* set the initial annealing temperature */
R ← current_time
Z ← 0 /* initialize the improvement counter */
while (R < time_limit) ∧ (Z < patience) do
    N ← select_node(Q, T) /* see Figure 10 */
    B ← select_bid(N, B) /* see Section 5.1.3 */
    N' ← expand_node(N, B) /* see Figure 11 */
    N'.v ← evaluate_node(N') /* see narrative in this section */
    insert(Q, N')
    if N'.v < N_opt.v then
        N_opt ← N'
        Z ← 0
    else
        Z ← Z + 1
    T ← T(1 − ε) /* the value ε is the annealing rate */
return N_opt
```

Figure 9: The Simulated Annealing algorithm used to evaluate bids.

search as an anytime search. If an agent is not able to process the bids fast enough, it might miss good deals and spend all its time processing bids instead of awarding contracts. A hostile agent could submit a huge number of bids to prevent another agent from accomplishing its task, although the Market Session can be configured to limit this.

A high level algorithm for our bid evaluation process is depicted in Figure 9. This is a simulated-annealing framework [29] with a finite queue (maximum length is *beam_width*). A number of complications are omitted, such as the fact that `expand_node` (see Figure 11) may fail to produce a node. This can happen, for example, because the selected bid has already been tried against the selected node, or because of an optional uniqueness test.

The simulated-annealing framework is a queue-based search, characterized by two elements, the annealing temperature T which is periodically reduced by a factor ε, and the stochastic node-selection procedure shown in Figure 10.

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

Figure 10: The node-selection algorithm.

Bid selection may be done by a variety of means. We describe several methods in Section 5.1.3. The selector is a plug-in component that can be configured in a number of ways. The simplest selector simply makes a random choice among bids that are not part of the current mapping. Others may focus on improving coverage, feasibility, or cost. In the next section we report on some experimental results using different bid selectors.

Node expansion (see Figure 11) is done by adding mappings of the selected bid to all the tasks covered by that bid, potentially discarding mappings of other bids that overlap the new bid. This will fail if the selected bid has already been used to expand the selected node, or because the bid was used to produce the current node or one of its parents within *tabu-length* generations (see [29] for a further explanation of the use of tabu lists). Not shown is the

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

Figure 11: The node-expansion algorithm.

fact that expansion can also fail if an attempt is made to unmap a singleton bid (a bid that provides the only possible mapping for some task).

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

- *coverage*: are all the subtasks covered? a bid that covers some not yet covered subtasks should be preferred over a bid that covers subtasks already covered.
- *feasibility*: is the current partial solution feasible? If the Critical Path algorithm finds any negative slack, the current partial solution is not feasible, since it violates some time constraint.
- *cost*: what is the total cost?
- *risk*: how risky is the solution? A low price bid might have a greater risk because the supplier has a poor reputation as reported by the Market's Better Business Bureau, or because its timing parameters may not provide sufficient slack to recover from failure. In general, risk factors include recovery cost, loss of value as the end date is delayed, cost of plan failure, and other factors.

When negotiation involves a large set of tasks with a rich set of temporal constraints, there is a tradeoff to be considered between guaranteeing that bids will compose feasibly (all

temporal constraints satisfied), or giving more flexibility in an attempt to reduce costs and potentially tighten up the schedule. If the latter approach is taken, the evaluation process becomes even more challenging. We have developed a set of evaluation criteria that can be traded off against each other, and the ability to add constraints (e.g. excluding a particular vendor, or ensuring that another vendor be given enough business to satisfy a long-term agreement).

An important component of bid evaluation is the determination of the expected schedule risk. Schedule risk is associated, among other things, with constraint-tightness, which is determined according to the slack available along each partial path through the task network.

Intuitively, the risk associated with constraint tightness is higher whenever accepting a particular set of bids will increase the probability of missing the goal deadline, or of missing the latest start time $b.t_{ls}(s)$ specified in the bid $b \in \mathcal{B}_s$ containing task s. This is similar to the problem addressed by the notion of *textures* in [12]. Our approach is to measure constraint tightness by *path*, where a path is a sequence of tasks starting at the beginning of the plan and extending toward the goal along successor relations. We use partial paths as well as complete paths for this calculation, because tasks can be constrained both by their precedence relationships and by the time windows specified in the bids. For example, in the task network in Figure 7, there are 9 paths, all starting with s_1, as follows:

s_1	$s_1 \prec s_3$	$s_1 \prec s_4$
$s_1 \prec s_2$	$s_1 \prec s_3 \prec s_5$	$s_1 \prec s_4 \prec s_5$
$s_1 \prec s_2 \prec s_6$	$s_1 \prec s_3 \prec s_5 \prec s_6$	$s_1 \prec s_4 \prec s_5 \prec s_6$

Along each path, we measure constraint tightness as the ratio of slack to expected duration, as in

$$k(path) = \frac{t_{lf}(s_{last}) - t_0 - \sum_{s \in path} d_e(s)}{\sum_{s \in path} d_e(s)}$$

where s_{last} is the last task on *path*, $d_e(s)$ is the expected duration of task s, and t_0 is the start time.

An more complete discussion of our approach to risk evaluation is contained in [6].

4.2.3 Accept Bids

After building the schedule, the Bid Manager sends bid acceptance messages to the vendors of accepted bids, specifying which parts of which bids are accepted. This completes bidding phase of the negotiation process.

4.3 Execution Manager

The Execution Manager is responsible for overseeing execution of the plan as contracted, and making decisions on how to respond when events do not proceed as expected. It receives the task assignments from the Bid Manager and, through the market, receives updates on plan execution from contracted vendors. It maintains a time map with tasks, vendor commitments, and temporal constraints among tasks. For each event, it must decide whether

to respond, and if so, whether to respond directly to a particular vendor, whether to re-bid a portion of the plan, or whether to re-plan and re-bid one or more subgoals of the plan.

The Bid Manager produces a set of task assignments, each of which includes one or more tasks from the plan, along with the contract data for execution of that task. Contract data include the task, the resources committed to carrying out that task, an agreed-upon price, an agreed-upon time window and temporal constraints, and the decommitment penalties.

All the activities of the Execution Manager revolve around the maintenance of the time map. The time map [10] can be thought of visually as a Gantt chart, decorated with contract data and temporal constraints among tasks. For each task the time map records an early start, a late finish time, the committed start time and duration, and the set of precedence constraints.

As time passes and the execution of the plan proceeds, the Execution Manager works in conjunction with the market session to drive the plan to completion. In general, the session is responsible for releasing tasks to the suppliers when their prerequisites are satisfied, and for assessing decommitment penalties when the parties fail to satisfy their commitments. In the process, the session forwards to the Execution Manager notifications of task release and task completion events. The Execution Manager is then responsible for making decisions and taking appropriate action in response to those notifications.

The market session maintains a *performance monitoring table*, which is essentially a stripped-down version of the time map maintained by the customer agent. It contains, for each contracted task, the winning bidder, the early start and late finish times, the contracted subset of successor tasks, and the decommitment penalty. The performance monitoring table is maintained by updating the expected release time for each successor task whenever the expected finish time of a task changes. When the slack value becomes critical ($slack(s) = 0$), or infeasible ($slack(s) < 0$) the customer is notified to enable it to take appropriate action.

Each supplier agent is also notified by the market session of task releases, and of changes to expected task release times for each task for which a bid has been awarded.

Following are the classes of events to which the Execution Manager must respond, and a brief outline of the response options.

Nominal Completion. No action is required.

Early Completion. If a critical path is affected, and if the value of the plan could be improved by changing the earliest start time for some task s, then the Execution Manager could request the vendor of s whether the schedule can be moved up, and for what cost. After evaluating the cost/benefit tradeoff, the Execution Manager will request schedule changes accordingly.

Vendor Decommitment. When a supplier decommits, the Execution Manager has three choices: (1) customer decommitment, (2) attempt to re-bid decommitted task(s), (3) attempt to re-plan and re-bid unsatisfied subgoal(s). The choice that is expected to maximize the profit is the one that will be made.

Missing Event. Completion events are considered missing if their failure to arrive triggers violation of a temporal constraint. This is considered non-performance on the part of

the vendor. The Execution Manager responds by notifying the market session of vendor non-performance.

Late Completion. A late completion event is one that occurs later than promised but does not violate temporal constraints. It is a configuration option whether to treat this as a missing event. If not, the Execution Manager responds by re-evaluating the critical path and notifying vendors of affected tasks of changed time windows. All such time window changes will result in tighter windows.

Notice of Late Completion. If a vendor wishes to extend a deadline, it must initiate a negotiation with the customer using a Notice of Late Completion, giving a new expected completion time and an updated bid. The Execution Manager must then choose whether to accept the updated bid, with the new time commitment, or whether to treat this event as a customer decommitment.

5 Experimental Results on Bid Evaluation

Bid evaluation is a critical part of the Customer agent's behavior, and it is clearly a costly combinatorial problem. We have put considerable work into building a search engine that performs well. It is important to characterize its performance because of the need to allocate time to it in the context of the overall negotiation process. We report on a set of experiments that give us confidence that effective bid evaluation can be performed in a reasonable time frame.

The first experiment compares the performance of our simulated-annealing search engine with a systematic search that is guaranteed to find all solutions. A solution is defined as a mapping of bids to tasks in which all tasks are covered by bids, and a feasible schedule can be composed, taking into account the precedence constraints in the plan and the time limits and quoted task durations in the bids. This experiment is somewhat limited in scope because of the obvious combinatorial limitations of the systematic search approach, but it does show that the simulated-annealing approach produces good results and seldom fails to find a solution if one exists.

The second experiment demonstrates the performance of the bid evaluation process using a set of increasingly informed bid selectors, and a selector that combines random and focused behaviors. It is clear from these results that the more focused selectors do not perform well on their own; they are not sufficiently "exploratory" to support the stochastic search approach. The "combined" selector is shown to perform well on relatively large problems.

5.1 Experimental Setup

The experimental setup includes three main components: a MAGNET Server as described in Section 6, a Customer agent that generates plans, requests quotes, and evaluates bids, and a simulated community of Supplier agents that generate and submit bids. The bid evaluation process is instrumented to measure the rate of improvement for various search strategies. Random variable seeds are controlled to ensure that different search strategies are presented

with exactly the same problems. Two seeds are used: one controls problem generation (the plan and the bids), and the second controls random number generation for the stochastic search. This guarantees that all search methods are run against the same set of problems.

5.1.1 Customer Agent: Construct and Issue a Request for Quotes

For these experiments, plans are randomly-generated task networks, with a number of controllable parameters, and the RFQs have a controllable amount of added slack to promote feasibility. Plan variables include:

1. Number of tasks.
2. Mix of task types. Task types are characterized by average duration, duration variability, average price, and price variability. Both duration and price are normally distributed, positive values.
3. Branch factor. This is the average number of precedence relationships per task. For example, if a particular task has one predecessor and one successor, then the branch factor for that task is 2.

Before issuing the RFQ, we compute the makespan for the entire plan, and expected early start times $s.t_{es}$ and late finish times $s.t_{lf}$ for each task in the plan, as specified in Section 4.2. RFQ variables include:

1. Total slack. This is the ratio of the time allowed for plan completion to the expected makespan of the plan.
2. Additional slack obtained by reducing task durations below their expected values. For these experiments, we simply multiplied each expected duration by a factor of 0.8 and re-ran the CPM algorithm. The only justification for this number is that it was the greatest reduction we found that did not cause most problems to have no feasible solutions. Other, more sophisticated methods of adding additional slack are possible, as outlined in Section 4.2.

5.1.2 Supplier Agent: Generate Bids

The supplier agent is a test agent that masquerades as an entire community of suppliers. Each time a new RFQ is announced by the Market, the supplier attempts to generate some number of bids.

Each bid represents an offer to execute some subset of the tasks specified in the RFQ. A price for the whole set is specified, along with prices for each task in the set. The ratio of the overall bid price to the sum of the individual task prices is referred to as the *discount* or the *premium* of the combination. In addition, the early start time, late finish time, and maximum duration are specified for each task. It is a requirement of the protocol that the time parameters in a bid are within the time windows specified in the RFQ. Task durations are normally distributed around the expected value used by the Customer, and the time

windows are randomly set to be smaller than the time windows specified in the RFQ and larger than the computed durations.

Bids are generated for random sets of contiguous tasks within the RFQ. Full details on the bid generation process are given in [8]. Previous work [39] has shown that the size of bids can have a significant impact on the difficulty of the search problem; intuitively, larger bids are harder to compose together because there is a higher likelihood of overlap.

5.1.3 Customer Agent: Evaluate Bids

Once the bidding deadline is past, the Customer evaluates the set of bids in an attempt to find a combination that provides coverage of all tasks, allows for a feasible schedule, and minimizes a combination of cost and risk. Since bids are exclusive OR (when multiple bids are submitted by the same agent, only one of them can be accepted) for each bid the Customer has the option of selecting which parts of the bid (if any) to accept. Bids for combinations might include a *discount* or a *premium* for accepting the whole bid.

The following bid-selection methods, used as the `select_bid` procedure in the algorithm in Figure 9, have been implemented and tested. Note that the feasibility and cost improvement methods have significant complexity costs associated with them, although this cost is incurred only once per node the first time the selector visits the node.

- *Random Bid, Random Bid Component*: Choose a bid or a bid component at random, and attempt to add it to the node. The ratio of bids to bid components is adjustable. This method is fast ($O(1)$) and promotes general exploration of the search space.

- *Coverage Improvement*: Choose a bid or bid component that covers a task that is not mapped in the node. The probability of choosing a bid component is equal to the coverage factor of the node. If a bid component is chosen, the coverage factor will increase; if a bid is chosen, the probability of improving coverage is always $P_{ci} > 0.5$. This method is also $O(1)$ if the set of unmapped tasks and the set of bids per task is stored.

- *Feasibility Improvement*: The mapping is scanned to find tasks that have negative slack $bid.t_{es} + duration_a > bid.t_{lf}$, are constrained by their bids rather than by predecessors or successors, and could be moved in a direction that would relieve the negative slack. They are sorted by their potential to reduce infeasibility, and saved. The untried bid or bid component with the highest potential to reduce infeasibility is chosen. Note that when a bid is chosen, there is no guarantee that it will not introduce other infeasibilities. The complexity of this method is $O(xy)$, where x is the number of tasks in the plan and y is the number of tasks in the mapping that meet the above improvement criteria, incurred once the first time a node is subjected to feasibility improvement, and $O(z)$, where z is the number of bids that could potentially be mapped to a task, the first time a feasibility improvement is attempted for a particular task on a node.

- *Cost Improvement*: Choose the (untried) bid or bid component that is responsible for the maximum positive deviation from the average price, and replace it with a lower-priced bid that covers at least the task with the highest positive cost deviation. The

first time this method is applied to a node, it has a complexity of $O(xy + z)$, where x is the number of bids mapped to a node, y is the number of components (tasks) in a bid, and z is the number of potential bids per task.

These selectors can be composed together and used to generate focused improvement for a given node. Available composite selectors include:

- *FeasCov*: If the node is infeasible, use the feasibility improvement selector; otherwise if it is not fully covered, use the coverage improvement selector; otherwise use the random selector.
- *CostFeasCov*: If the cost of the covered portion of the node is above average, attempt to reduce its cost; otherwise use the *FeasCov* selector.
- *Combined*: Run the *Random* selector as long as it produces improvement, then switch to *Feasibility Improvement* until that fails to produce improvement, then switch back to *Random*, then to *Coverage Improvement*, then back to *Random*, then to *CostFeasCov*, and finally back to *Random*.

5.2 Comparing Systematic and Stochastic Search

Our first experimental goal was to determine how well the simulated annealing search performed with respect to a known optimal reference. For that purpose, we constructed an alternate search engine that generates all feasible combinations of bids and bid components in order to be guaranteed of finding optimal solutions. Because of bid overlap and feasibility issues, no more efficient method is known that will provide such a guarantee. Its structure is similar to the method reported in [39] with the addition of a feasibility test.

The test problem for this experiment is necessarily small, because of the long run times of the systematic search engine. We generated 20 random problems with 10 tasks and 11 bids each. The "branch factor" that controls the density of precedence relationships was 2.4, and the average bid size was 2.72 tasks. Overall schedule slack was set to 1.4, and task durations were set to 70% of expected values to open up the time windows in the RFQ.

The summary data for this experiment is in the table below. Of the 20 problems, solutions (feasible mappings that covered all tasks) were found for 9 problems. The others either lacked coverage (in 7 of the 20 runs there was at least one task for which no bid was submitted) or no feasible combinations existed (4 cases). The node counts and the solution evaluations are the mean for the cases where solutions were found. The data for the Stochastic 1 and Stochastic 3 trials are normalized to account for the missing solution.

	Systematic	Stochastic 1	Stochastic 2	Stochastic 3
Nodes Generated	115480	2171	2205	1323
Covered & Feasible	46916	435	448	219
Best Solution Eval.	7960	8073	7998	8073
Solutions Found	9	8	9	8
Run Time (min.)	242	9.8	9.4	6.2

The 4 trials used identical plans and bid sets. The search engine parameters were set up as follows:

Systematic: Systematic search with problem setup as described above.

Stochastic 1: Simulated-annealing search, initial temperature 0.35, reduced by 0.95 every 100 iterations, patience factor (number of iterations without improvement) 100. The stopping criterion for the stochastic search is either timeout (no timeout was used in this trial) or failure to improve for a number of iterations equal to the patience factor. The bid selector is the Combined selector, as described in the previous section.

Stochastic 2: Setup as in Stochastic 1, except that a uniqueness test is added to prevent identical nodes from being evaluated and added to the search queue.

Stochastic 3: Setup as in Stochastic 2, except that the patience factor was reduced to 50.

The conclusion is that the simulated-annealing search engine performs well, with solution quality within 2% of the systematic search at radically reduced run times. It is also clear that it needs to be tuned to avoid missing solutions.

5.3 Stochastic Search: Comparing Bid Selectors

In this study, we are attempting to learn whether the simulated-annealing search technique can be effectively applied to large bid-evaluation problems. The results show that the *Combined* selector outperforms all the others and can be used effectively on moderate-sized problems, and that the more focused methods alone are ineffective, even if used with high annealing temperatures. It seems that excess focus on improvement leads to faster improvement early on, at the cost of a lower likelihood of finding a solution that satisfies all constraints.

In order to probe a range of problem complexity factors, we ran the *Random*, *Cov*, *FeasCov*, and *Combined* selectors against two different problem types of the same size but different levels of complexity. Both types contain 50 tasks and 100 bidders, and each problem set is generated with the same random number sequence. In the *small-bid* problem, the average bid size (number of tasks included in a discounted bid) is 5, and in the *large-bid* problem, the average bid size is 15. Earlier work [39] has shown that this difference has a significant impact on the search difficulty due to the greater probability of overlap among bids.

Figure 12 shows the improvement curves for the four bid selectors on the *small-bid* problem, and Figure 13 shows improvement curves for the same selectors on the *large-bid* problem. Error bars show $\frac{\sigma}{\sqrt{n}}$ where σ is the standard deviation across runs, and n is the number of runs. The *Combined* selector clearly gives the best overall performance, both in terms of solution quality and in terms of consistency.

The following table shows the number of acceptable assignments found for the *small-bid* and *large-bid* problems. The table shows how effective the four selectors were at finding solutions that satisfied all constraints. The actual number of such solutions is not known. Again, we see the advantage of the *Combined* selector, which uses random selection to generate sets of candidates, and then switches to more focused selectors to clean up.

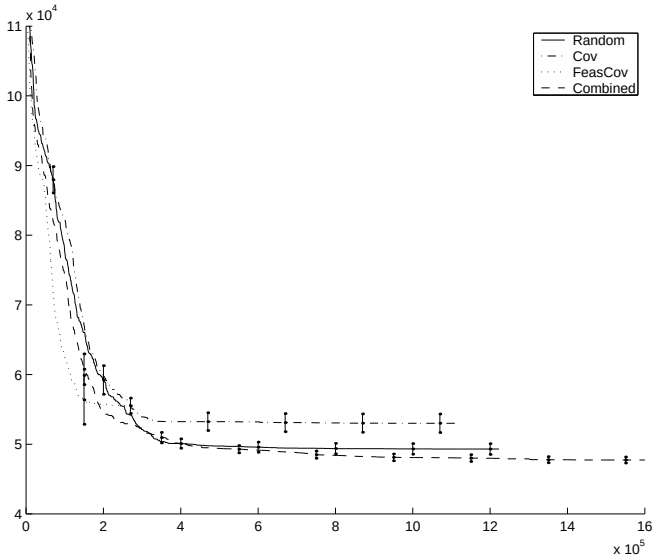

Figure 12: Improvement curves for the *small-bid* problem. Averages are shown for 20 runs.

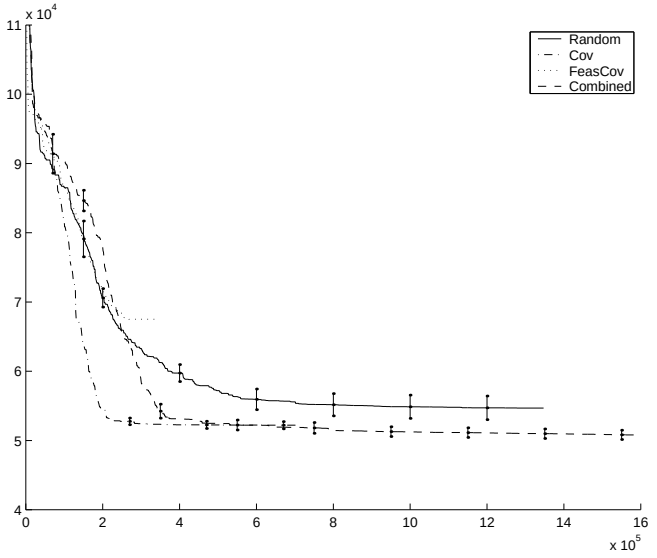

Figure 13: Improvement curves for the *large-bid* problem. Averages are shown for 20 runs.

Selector	small-bid problem	large-bid problem
Random	2	2
Cov	3	0
FeasCov	2	0
Combined	6	1

In Figure 14, we explore the effect of raising the annealing temperature on the performance of the selectors. The experiments described earlier are all run with an initial annealing temperature of 0.3. We see that raising the annealing temperature does not improve performance, and the focused selectors do not perform any better at higher temperatures.

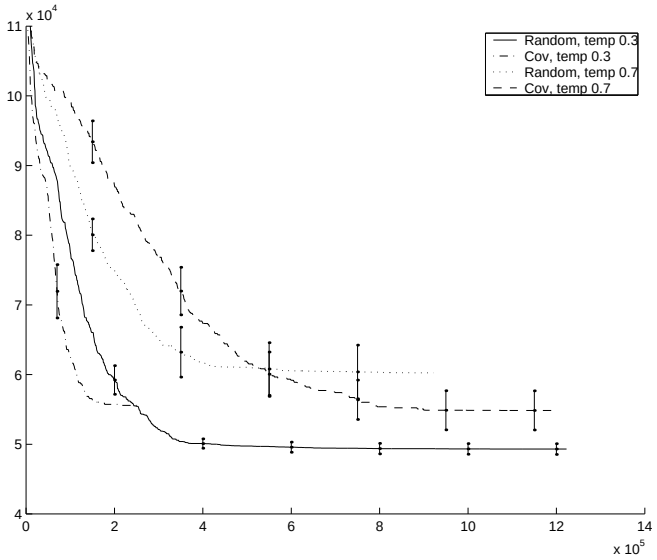

Figure 14: Improvement curves for two different annealing temperatures.

6 Implementation

Figure 15 is a UML class diagram that shows the principal interfaces of a MAGNET server. In addition to the basic Exchange, Market, Session structure introduced in section 3.1, we add the additional notion of a *Participant*, which acts as a server-side proxy for an individual agent. A Participant will have a *ParticipantRole* instance for each Session it has joined: a *Customer* for those Sessions in which it is playing the Customer role, and a *Supplier* for those Sessions in which it is playing the Supplier role. The ParticipantRole objects provide the necessary role-specific interfaces to the Sessions, filtering content and enforcing protocol rules. The principal Agent-visible content is contained in a *SessionContext* object, which is accessible through the *getSessionContext* method in the ParticipantRole. The MarketHome, SessionHome, and ParticipantHome interfaces are factory/finders for the Markets, Sessions, and Participants respectively.

The server is highly scalable. Markets and their Sessions can be distributed separately from an Exchange, and it is possible for a Market to be registered with multiple Exchanges, although this feature has not been implemented. Each Exchange and each Market is built on an underlying database, giving it long-term persistence and transactional robustness.

When an agent wishes to submit an RFQ to a Market, the sequence of events is approximately as follows:

1. If the agent has not yet registered with the Exchange, it must do that in order to obtain a security context, which is required for all other operations.

2. The agent logs in to the Exchange, obtaining a reference to its Participant interface and its security context.

3. The agent chooses a Market in which to run its negotiation Session.

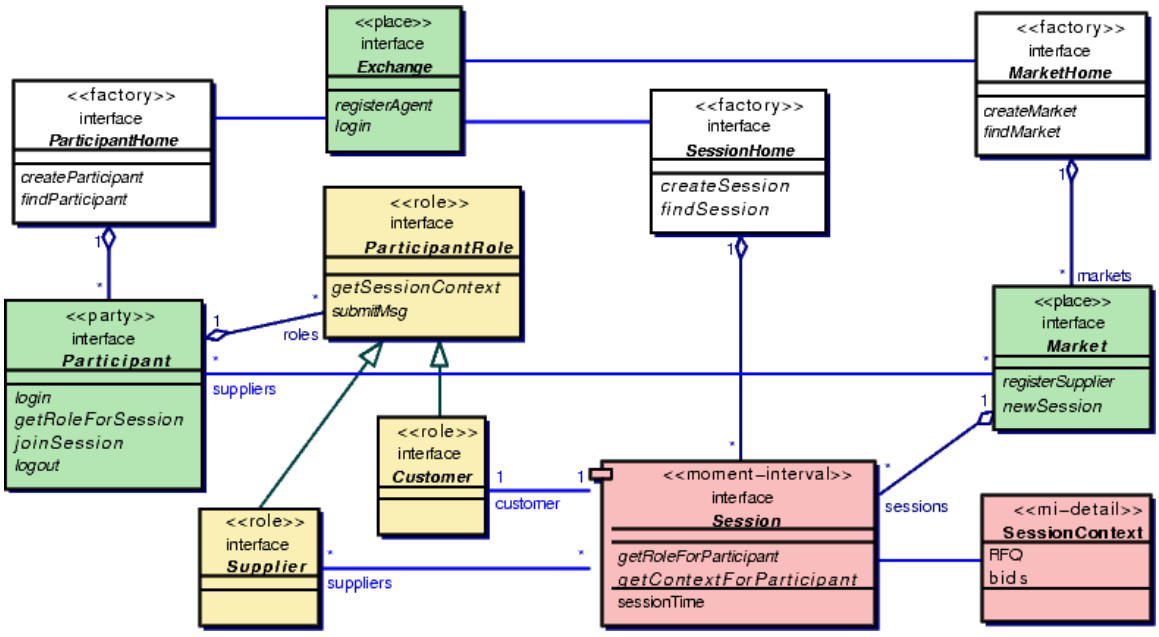

Figure 15: Design of the MAGNET Server

4. The agent requests a new Session from the Market. This gives the agent a Customer role interface with which to interact with the Session. This also connects the agent with the protocol event stream (normally via a callback interface) produced by the role object.

5. The agent composes its RFQ and submits it to the Session via its role interface.

6. The Market inspects the new RFQ and forwards it to Suppliers who have registered an interest in its contents.

7. Suppliers who wish to participate in the negotiation join the new Session in order to obtain a Supplier role interface. They may then retrieve the SessionContext for that session, which will contain the RFQ.

8. Negotiation proceeds between the Customer and Supplier agents, each submitting protocol messages and receiving notifications through their respective role interfaces.

The current MAGNET server is implemented in Java as described above, with persistent state that allows shutdown and recovery without loss of data. In addition, an agent framework has been constructed that provides a local interface to a remote server, including a callback mechanism for receiving protocol events. Simple agents have been constructed on this framework, including the test agents described in this paper, as well as a user-interface customer agent that allows a user to interact directly with the server, and through the server with other agents. We are currently working on integrating the user interface with the search engine described here and automated protocol processing to support a mixed-initiative interaction model.

7 Related Work

Markets play an essential role in the economy [1], and market-based architectures are a popular choice for multiple agents (see, for instance, [5, 30, 42, 47] and our own MAGMA architecture [44]). Most market architectures limit the interactions of agents to manual negotiations, direct agent-to-agent negotiation [35, 11], or some form of auction [49].

Auctions are becoming the predominant mechanism for agent-mediated electronic commerce [16]. AuctionBot [49] and eMEDIATOR [34] are among the most well known examples of multi-agent auction systems. They use economics principles to model the interactions of multiple agents [47]. Auctions are not the most appropriate mechanism for the business-to-business transactions we are interested in, where convenience of scheduling, reputation, and maintaining long term business relations are often more important than cost.

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

Rosenschein and Zlotkin [32] showed how the behavior of the agents can be influenced by the set of rules that the system designers choose for the agents' environment. In their study the agents are homogeneous and there are no side payments. In other words, the goal is to share the work, not to pay for work. They also assume that each agent has sufficient resources to handle all the tasks, while we assume the contrary.

Sandholm's agents [36, 35] redistribute work among themselves by a contracting mechanism. Sandholm considers agreements involving explicit payments, but he also assumes that the agents are homogeneous – they have equivalent capabilities, and any agent can handle any task. Our agents are heterogeneous, and decide what tasks to handle.

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 [40]. Kuokka and Harada [22] describe an agent application whereby potential producers and consumers of information send KQML messages describing their capabilities and needs to an intermediary called a matchmaker. Sycara et al [41] present a language 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 determination of winners of combinatorial auctions [25] is hard. Dynamic programming [33] works well for small sets of bids, but does not scale and imposes significant restrictions on the bids. Sandholm [34] relaxes some of the restrictions and presents an algorithm for optimal selection of combinatorial bids, but his bids include only cost.

Our setting is more general. MAGNET agents have the ensure the scheduling feasibility of the bids they accept. They also have to assess their own preferences and risk tolerance. In our bid evaluation algorithm the Customer agent evaluates the different choices using a

measure of risk in addition to just cost.

We have chosen to use a simulated annealing framework for bid evaluation. Since the introduction of iterative sampling [23], a strategy that randomly explores different paths in a search tree, there have been numerous attempts to improve search performance by using randomization. Randomization has been shown to be useful in reducing the unpredictability in the running time of complete search algorithms [15].

A variety of methods that combine randomization with heuristics have been proposed, such as Least Discrepancy Search [18], heuristic-biased stochastic sampling [4], and stochastic procedures for generating feasible schedules [27], just to name a few. The algorithm we presented is based on simulated annealing, and as such combines the advantages of heuristically guided search with some random search. Our experimental results show that additional benefit can be obtained by using domain-specific heuristics when deciding how to expand a node. This combined with the basic simulated annealing framework produces good results in a short time frame.

Our goal is to automate the scheduling/execution cycle of a single autonomous agent that needs the services of other agents to accomplish its task. Pollack's DIPART system [28] and the Multiagent Planning architecture (MPA) [48] assume multiple agents that operate independently but all work towards the achievement of a global goal. Our agents are trying to achieve their own goal and to maximize their profit; there is no global goal.

Various classes of scheduling problems have been considered in the Operations Research literature [20, 24, 13]. Many interesting scheduling problems are computationally intractable, and numerous heuristic approaches have been described [17, 31]. Much of the recent work in scheduling has focused on the problem of maintaining or updating an existing schedule in the face of changes [50, 38]. All of these systems are concerned with determining schedules for individual resources; the assumption is that there is some set of tasks to be done, and some set of resources available to do those tasks, and the problem is to find an optimal or near-optimal assignment of tasks to resources over time. MAGNET Customer agents are not resource-limited; these scheduling approaches will work for the supplier agents, but they are not well suited for the problems that the Customer agent must solve.

The notion of using an explicit time-map to support reasoning about events, actions, states, and causality over time was first described in [10] and further elaborated in [3]. Schwalb, Kask, and Dechter [37] describe a formal system for reasoning about time and events, including another form of a time map called a Conditional Temporal Network.

The problem faced by the Customer agent in our design is that of monitoring a plan and its schedule, and finding ways to repair it when it is broken. Muscettola [26] and Tate et al [43] advocate combining the planning and scheduling problems to deal with this issue. Muscettola's approach is based on maintaining state information over time for a relatively fixed set of resources, and so it is not directly applicable to our work.

8 Conclusions and Future Work

In this paper we have brought together ideas from recent work in market architectures for electronic commerce, and work in multi-agent contracting protocols. We have presented a generalized market architecture that provides support for a variety of transaction types,

from simple buying and selling to complex multi-agent contract negotiations. We have also presented a protocol that takes advantage of the services of the market. Our market architecture is organized around three basic components: the exchange, the market, and the session. We have shown how the existence of an appropriate market infrastructure can add value to a multi-agent contracting protocol by controlling fraud and discouraging counterspeculation.

Bid Evaluation is an important, difficult, combinatorial problem. We have shown that a flexible, tunable Simulated Annealing search framework can be used to solve this problem. We have begun to characterize the temporal performance of our search engine, in order to support the time-allocation decisions an agent must make during the bidding cycle.

We have begun implementing software-agents to interact within MAGNET, and intend to benchmark their performance against human participants as well as each other. We believe that the most productive, realistic approach to agent design in a commercial contracting environment will be a mixed-initiative approach, in which the agent's responsibility is to act as a decision-support tool and intermediary for a human decision maker. We are also exploring how modifying the parameters of the market and session protocols can effect the performance of the system. Eventually, we would like to open our testbed to outside participation over the Internet.

References

[1] Yannis Bakos. The emerging role of electronic marketplaces on the Internet. *Comm. of the ACM*, 41(8):33–42, August 1998.

[2] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *Proc. of the 11th Joint Conf. on Artificial Intelligence*, volume 2, pages 979–984, Detroit, MI USA, August 1989.

[3] Mark S. Boddy. Temporal reasoning for planning and scheduling in complex domains: Lessons learned. In Austin Tate, editor, *Advanced Planning Technology*, pages 77–83. AAAI Press, Menlo Park, CA, 1996.

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

[5] Anthony Chavez and Pattie Maes. 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, April 1996.

[6] John Collins, Corey Bilot, Maria Gini, and Bamshad Mobasher. Mixed-initiative decision support in agent-based automated contracting. In *Proc. of the Fourth Int'l Conf. on Autonomous Agents*, June 2000. (to appear).

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

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

[9] John Collins, Ben Youngdahl, Scott Jamison, Bamshad Mobasher, and Maria Gini. A market architecture for multi-agent contracting. In *Proc. of the Second Int'l Conf. on Autonomous Agents*, pages 285–292, May 1998.

[10] Thomas L. Dean and Drew V. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[11] Peyman Faratin, Carles Sierra, and Nick R. Jennings. Negotiation decision functions for autonomous agents. *Int. Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1997.

[12] Mark S. Fox, Norman Sadeh, and Can Baykan. Constrained heuristic search. In *Proc. of the 11th Joint Conf. on Artificial Intelligence*, volume 1, pages 309–315, Detroit, MI USA, August 1989.

[13] M. R. Garey, R. L. Graham, and D. S. Johnson. Performance guarantees for scheduling algorithms. *Operations Research*, 26(1):3–21, January 1978.

[14] Judith Gebauer and Arie Segev. Assessing internet-based procurement to support the virtual enterprise. *Virtual-Organization.net Newsletter*, 2(3), September 1998.

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

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

[17] Othar Hansson and Andrew Mayer. DTS: A decision-theoretic scheduler. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, pages 371–388. Morgan Kaufmann Publishers, San Francisco, CA, 1994.

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

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

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

[21] D. M. Kreps. *A Course in Microeconomic Theory.* Princeton University Press, 1990.

[22] Daniel Kuokka and Lawrence Harrada. On using KQML for matchmaking. In *Proc. 3rd Int'l Conf on Information and Knowledge Management*, pages 239–245, 1995.

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

[24] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, January 1978.

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

[26] Nicola Muscettola. HSTS: Integrating planning and scheduling. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 6, pages 169–212. Morgan Kaufmann, San Francisco, CA, 1994.

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

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

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

[30] J. A. Rodriguez, P. Noriega, C. Sierra, and J. Padget. 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, April 1997.

[31] Y. Ronen, D. Mosse', and Martha E. Pollack. Value-density algorithms for the deliberation scheduling problem. *SIGART Bulletin*, 7(2), 1996.

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

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

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

[35] Tuomas W. Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents*. PhD thesis, University of Massachusetts, 1996.

[36] Tuomas W. Sandholm and Victor Lesser. On automated contracting in multi-enterprise manufacturing. In *Distributed Enterprise: Advanced Systems and Tools*, pages 33–42, Edinburgh, Scotland, 1995.

[37] Eddie Schwalb, Kalev Kask, and Rina Dechter. Temporal reasoning with constraints on fluents and events. In *Proc. of the Twelfth Nat'l Conf. on Artificial Intelligence*, Seattle, WA, August 1994. AAAI.

[38] Stephen F. Smith. OPIS: A methodology and architecture for reactive scheduling. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 8, pages 29–66. Morgan Kaufmann, San Francisco, CA, 1994.

[39] Erik Steinmetz, John Collins, Maria Gini, and Bamshad Mobasher. An efficient algorithm for multiple-component bid selection in automated contracting. In *Agent Mediated Electronic Trading*, volume LNAI1571, pages 105–125. Springer-Verlag, 1998.

[40] Katia Sycara, Keith Decker, and Mike Williamson. Middle-agents for the Internet. In *Proc. of the 15th Joint Conf. on Artificial Intelligence*, pages 578–583, 1997.

[41] Katia Sycara, Matthias Klusch, Seth Widoff, and Jianguo Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record (ACM Special Interests Group on Management of Data)*, 28(1):47–53, March 1999.

[42] Katia Sycara and Anandeep S. Pannu. The RETSINA multiagent system: towards integrating planning, execution, and information gathering. In *Proc. of the Second Int'l Conf. on Autonomous Agents*, pages 350–351, 1998.

[43] Austin Tate, Brian Drabble, and Richard Kirby. O-plan2: An open architecture for command, planning, and control. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 7, pages 213–240. Morgan Kaufmann, San Francisco, CA, 1994.

[44] Maxsim Tsvetovatyy, Maria Gini, Bamshad Mobasher, and Zbigniew Wieckowski. MAGMA: An agent-based virtual market for electronic commerce. *Journal of Applied Artificial Intelligence*, 11(6):501–524, 1997.

[45] H. R. Varian. Economic mechanism design for computerized agents. In *USENIX Workshop on Electronic Commerce*, New York, NY, July 1995.

[46] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

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

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

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

[50] Monte Zweben, Brian Daun, Eugene Davis, and Michael Deale. Scheduling and rescheduling with iterative repair. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 8, pages 241–256. Morgan Kaufmann, San Francisco, CA, 1994.