Bid Evaluation and Selection in the MAGNET Automated Contracting System

Erik Steinmetz, John Collins, Scott Jamison, Rashmi Sundareswara,
Bamshad Mobasher, and Maria Gini

Department of Computer Science and Engineering
University of Minnesota

Abstract. We present an approach to the bid-evaluation problem in a system for multi-agent contract negotiation, called MAGNET. The MAGNET market infrastructure 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. One responsibility of a customer agent in the MAGNET system is to select an optimal bid combination. We present an efficient anytime algorithm for a customer agent to select bids submitted by supplier agents in response to a call for bids. Bids might include combinations of subtasks and might include discounts for combinations. In an experimental study we explore the behavior of the algorithm based on the interactions of factors such as bid prices, number of bids, and number of subtasks. The results of experiments we present show that the algorithm is extremely efficient even for large number of bids.

1 Introduction

The combination of electronic commerce and autonomous intelligent agents has the potential to deliver enormous economic benefits. Primitive examples are already being deployed on the Internet, in the form of automated shopping agents [8] and auction services [17,26]. More complex economic activities remain outside the reach of the current generation of automated agents.

The overall research goal of the MAGNET project [6,7] is to develop a semantic model for the integration of planning, contracting, scheduling, and execution in a multi-agent market domain, such as the Internet. In particular, we are interested in how an agent that has a goal to satisfy can construct a plan, issue a call for bids to other self-interested agents, award contracts, and monitor their execution. We call this process *Plan Execution by Contracting*.

MAGNET includes a market infrastructure and a set of agents that can make use of this environment to carry out Plan Execution by Contracting activities. The market infrastructure provides an environment with explicit support for complex agent interactions. The market acts as a trusted third party to reduce opportunities for fraud and misrepresentation. It also manages and enforces the negotiation protocol between agents, from the negotiation and contracting phases through the full cycle of contract commitment, execution, and settlement.

All the agents in the MAGNET environment are assumed to be self-interested. In other words, they exhibit limited rationality in the sense that they will do what is in their own best interests within the limits of their reasoning capabilities. Agents are also heterogeneous; they are not assumed to have the same capabilities, nor do they necessarily embody similar decision processes. In general, they are motivated to engage in contracting behavior because they do not have direct access to the resources needed to execute their plans.

The main focus of this paper is a bid-evaluation mechanism for MAGNET agents. We start by providing an overview of the MAGNET market architecture and a negotiation protocol for Plan Execution by Contracting, and then proceed to discuss bid evaluation. Because the evaluation of bids must take into account plan feasibility as well as cost factors, straightforward auction mechanisms are inadequate for the MAGNET domain. We describe an anytime bid-evaluation algorithm that attempts to find the lowest-cost feasible plan, within the limits of available time and computing resources. Finally, we describe how our work relates to other efforts in the general area of automated negotiation and contracting.

2 The MAGNET Architecture

The MAGNET architecture is a distributed set of objects that can support electronic commerce in a variety of domains, from the simple buying and selling of goods to situations that require complex multi-agent negotiation and contracting. The fundamental elements of this architecture are the *exchange*, the *market*, and the *market session*, as outlined below.

2.1 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, or 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.

2.2 Markets

Each *Market* within an exchange is a forum for commerce in a particular commodity or business area. We envision markets devoted to banking, publishing and printing, construction, transportation, industrial equipment, electronic assembly, 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.

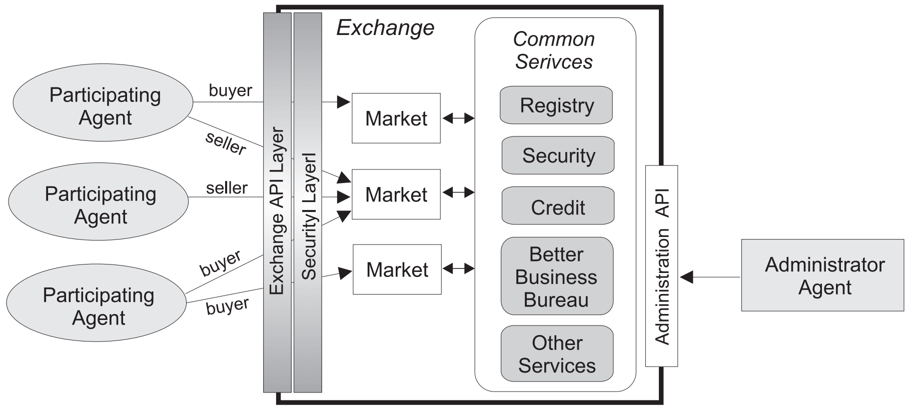

Fig. 1. The Structure of an Exchange

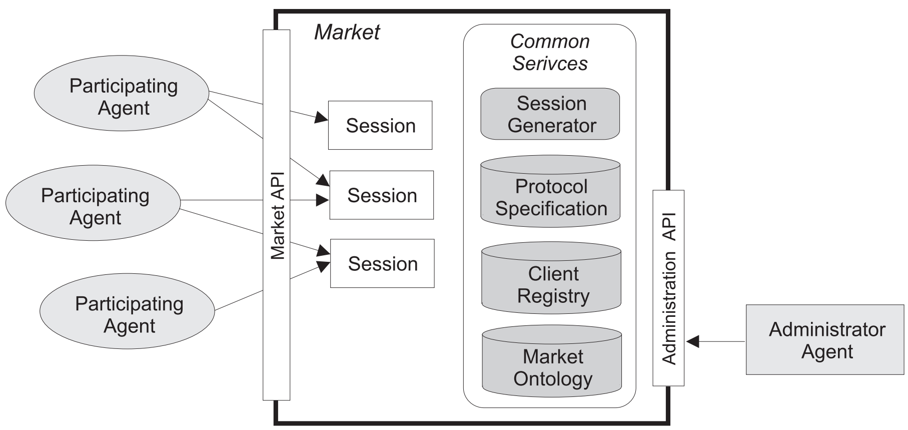

Fig. 2. The Structure of a Market within the Exchange

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 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.

- A *Protocol Specification* that formalizes the types of negotiation supported within the market. Within a planning-oriented market domain, these specifications include limits on parameters of the negotiation protocol, such as the maximum decommitment penalty, whether bids can be awarded before the bid deadline, etc.
- A *Registry* of market clients who have expressed interest in doing business in the market. Entries in this registry would include the identity of a client, a catalog (or a method for accessing a catalog) of that client's interests, products or capabilities, which can be used to locate clients to meet requests for new session participants, and a client agent that is empowered to negotiate contracts on behalf of the supplier. Client catalogs are required to express their interests and offerings in terms of the market's ontology.

2.3 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 construction-oriented market, if an agent wishes to build a new house, it initiates a session and issues a call-for-bids. The session extends from the initial call-for-bids 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 possibly a set of related contracts). The session mechanism ensures continuity of partially-completed transactions, protects against fraud by verifying the identity of agents, limits counterspeculation by enforcing negotiation rules, 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 clients*. A session can be initiated either for the purpose of buying or selling, depending on the type of market. In the above example of building a house, the initiating agent was the buyer or customer, and the other participants would be sellers or suppliers, whether they were supplying materials, labor, advice, credit, or other services. 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, while the house-building session described above would be closed once the contracts were let. The market maintains a list of open sessions which may be accessed (and potentially joined) by participating agents.

Figure 3 shows the structure of a session. Two APIs are exposed, one for the session initiator and one for session clients.

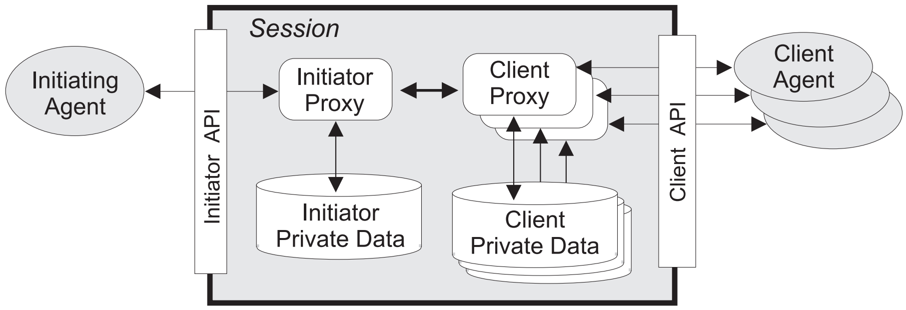

Fig. 3. The Structure of a Market Session

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 could be instantiated at different locations in the network to maximize performance.

3 The MAGNET Protocol

In this section we briefly describe a protocol that supports the Plan Execution by Contracting model. As outlined in the interaction diagram in 4, the negotiation portion of the protocol is a finite 3-step process that begins when a customer agent initiates a session and issues a Call For Bids. This diagram does not deal with decommitment or settlement.

The Plan Execution by Contracting protocol begins after the session has been initiated by a customer agent: the customer issues a call-for-bids, suppliers reply with bids, and the customer accepts the bids it chooses with bid-accept messages.

Another set of messages, including release, completion, and decommitment, are used to manage the progress of the plan once bids have been awarded. We have avoided the need for open-ended negotiation by means of bid break downs and a time-based decommitment penalty as described below.

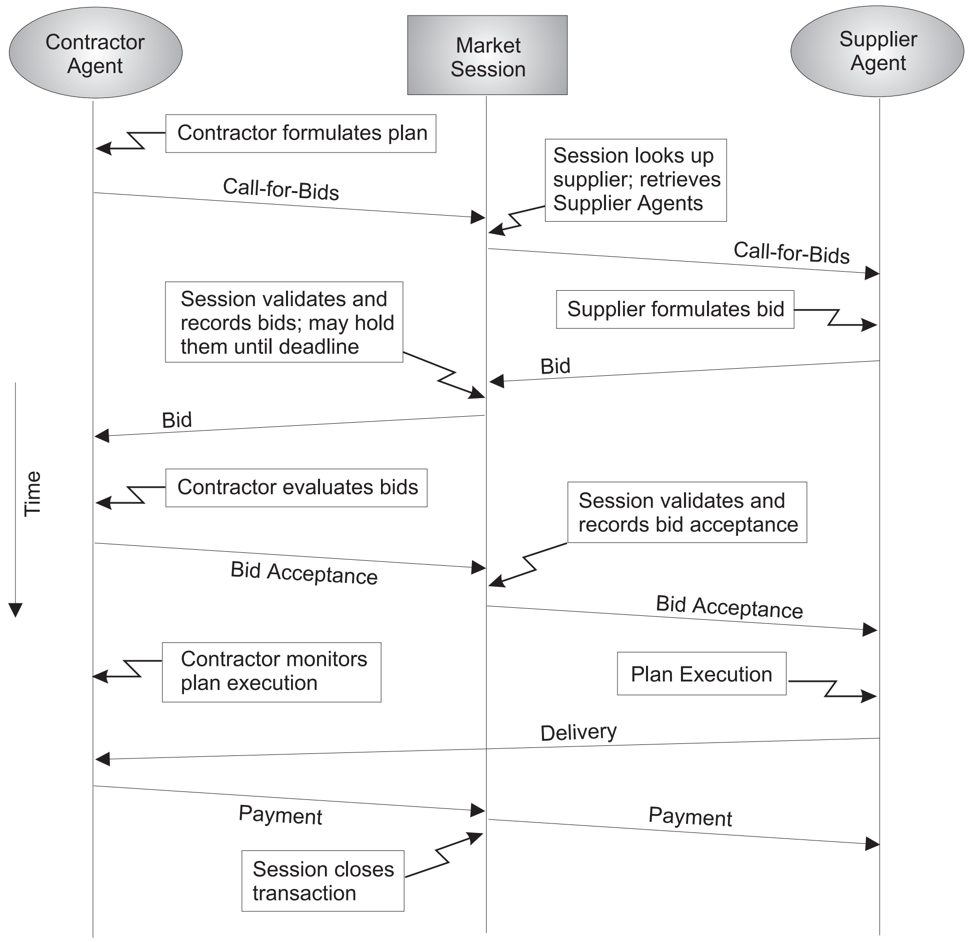

Fig. 4. A Typical Session-Mediated Negotiation

3.1 Call for bids

Once the customer has developed a plan of subtasks chosen from the market's ontology, it will send a call-for-bids message. The call-for-bids message will include, for each subtask listed, a time window during which the work must be done. The call-for-bids message will also include, among other information:

1. a bid deadline, or the time by which the suppliers must respond with bids,
2. the time at which the customer will begin considering the bids,
3. the earliest time at which bid acceptances will be sent,
4. penalty functions for each subtask, which will be assessed against the supplier if the supplier commits to work, but fails (or decides not) to do it. These penalty functions are piecewise-linear functions of time that are intended to

encourage suppliers to perform the work they commit to. If a supplier is unable to perform, the increasing value of the penalty function encourages it to explicitly decommit as early as possible.

This call-for-bids message, once created, is passed to the market session, which makes it available to all of the appropriate suppliers (those who are registered with the market, and are able to perform the necessary tasks.)

In this sense, the call-for-bids message is public, while all of the remaining messages are private. Before forwarding it, the market session may check the message to make sure that it conforms to all market and exchange rules which may exist.

3.2 Bidding

Each supplier will inspect the call-for-bids, and will decide whether or not it should respond with a bid, according to its resources, time constraints, and knowledge of the work to be done, according to the catalog of services provided by the market agent.

If it chooses to respond, it will send a bid message, which will be private (i.e. other suppliers will not see the contents of the bid). This bid message can include a combination of subtasks, which must be a subset of the subtasks listed in the call-for-bids. The content and number of bid messages will be monitored and may be recorded by the market session, before they are validated and forwarded to the customer.

In the bid, the supplier must indicate the cost (to the customer), the time window, and the estimated duration of the work for the whole subtask combination, and this same data for each of the separate subtasks (please see the explanation for this in the next section). The bid-accept deadline must also be included, as well as a penalty function for each subtask which the customer will have to pay if it commits to giving this supplier the work but then decides to decommit. This penalty function will have the same structure as the supplier penalty function.

Each supplier can send multiple bids for each call-for-bids, each including different costs and time windows, but each supplier will be awarded only one bid combination (or part of one). This is to enable the supplier to send many bids, but not over commit itself.

This bid is a commitment by the supplier to do work listed in the bid, should the customer accept it. If the supplier sends no bid message before the customer's bid deadline, the customer will assume that the supplier has decided not to send a bid for this particular call for bids. Thus, rejection is passive.

3.3 Bid acceptance

Having received the bids, the customer must decide which of the bids to accept, using knowledge about the bids, the task and subtask values, its own time constraints and the bidder (perhaps provided by the market agent). After completing this process, the customer must decide to do one of three things for each bid that it has received:

1. accept the whole bid,
2. accept a subset of the subtasks in the bid, or
3. reject the bid (passive rejection).

The motivation for these choices is to make open-ended negotiation unnecessary. If no acceptable set of bids together would cover every subtask to the satisfaction of the customer, then the customer can avoid negotiation because it knows how the supplier will break down the costs of the accepted subtasks, should it become necessary for the customer to accept a subset of the original bid combination.

This scheme in conjunction with the time-based decommitment penalty functions makes it possible to avoid open-ended negotiation without loss of generality.

The bid-accept message will be sent through the market session, which will verify, validate and time-stamp it before forwarding it to the customer. Note that either of the first two choices are commitments to give the supplier the work and at the point in time that this message is sent (according to the market session's time stamp), both the supplier and the customer penalty functions will be set into effect.

A failure to send a bid-accept message means the customer is rejecting the supplier's bid.

Once commitments have been made, an agent may determine that it cannot do the tasks it has committed to, or that it would disadvantageous to do so. In these situations, the agent must send a decommitment message to the other agent, describing what parts of its commitment it will not be satisfying. Included in the decommitment messages will be an acknowledgment of the penalty that the agent will be paying as a result of the decommitment.

3.4 Release

As the plan progresses, Release messages are used to inform supplier agents that they may begin work on portions of the plan for which they have been awarded bids. Failure to release prior to the suppliers latest start time constitutes decommitment on the part of the customer, and a penalty will be assessed by the Session.

3.5 Decommitment

Once bids are awarded, either party may choose to decommit and pay a penalty. The ability to decommit makes this a "leveled commitment" protocol. This is a requirement in many real-world contracting domains, and Sandholm and Lesser [21] have shown that the ability to decommit permits agreements to be reached in situations where no agreement would otherwise be possible.

Decommitment is only valid prior to delivery, and the penalty is not discounted in the case where a discounted multi-element bid was awarded.

3.6 Delivery and acceptance

The protocol is completed with messages that signify delivery by the supplier and acceptance of delivery by the customer. Failure to deliver prior to the deadline agreed to in the bid constitutes supplier decommitment, and the supplier will be assessed the decommitment penalty by the Session. For present purposes, we will assume that settlement is outside the scope of the system.

4 An Algorithm for Bid Evaluation

In this section we consider the specific problem of evaluating bids in a Plan Execution by Contracting situation. In general it is not enough to merely compare prices, because the set of bids accepted must constitute a complete and feasible plan. We have chosen a local improvement search [22,27] over a constructive search for three reasons:

- There is a straightforward mechanism for constructing a baseline feasible solution.
- The time-dependent nature of the negotiation protocol requires that the search be completed within a fixed period of time. Boddy and Dean [3] have characterized this type of search as an anytime search. In [2], Boddy has further characterized the requirements for anytime problem solving using performance profiles.
- Since the search space for this problem is well-structured, a systematic, domain-specific search algorithm such as the one we propose here appears more suited than the generic methods described in [16].

We consider a typical contracting situation in which the customer's call-for-bids is comprised of a group of subtasks. We use bid break-downs to avoid open-ended negotiation among agents, but for simplicity, we do not consider temporal factors such as bid deadlines or time-based decommitment penalties. Accordingly, a bid by a supplier is a subset of these subtasks with an associated cost or price for the whole bid.

In addition, each bid includes a cost for individual subtasks that make up the bid. The bid cost may represent a *discount* over the sum of the costs of individual subtasks contained in the bid. To satisfy a subtask the customer agent has the option of choosing the whole bid from a given supplier, or selecting individual bid elements from various suppliers.

A typical contracting situation is depicted in Figure 5, where the customer agent has issued a call-for-bids comprised of four subtasks S_1, S_2, S_3, and S_4. Suppliers have submitted 3 bids, each containing a subset of these subtasks. Finally, the customer, after evaluating the bids, has accepted parts of bids 1 and 3 and all of bid 2. In this case, the customer would pay the full price for subtasks S_1 and S_4 as specified by bids 1 and 4, respectively. However, subtasks S_2 and S_3 may have been obtained at a discount price since the customer has accepted the complete bid.

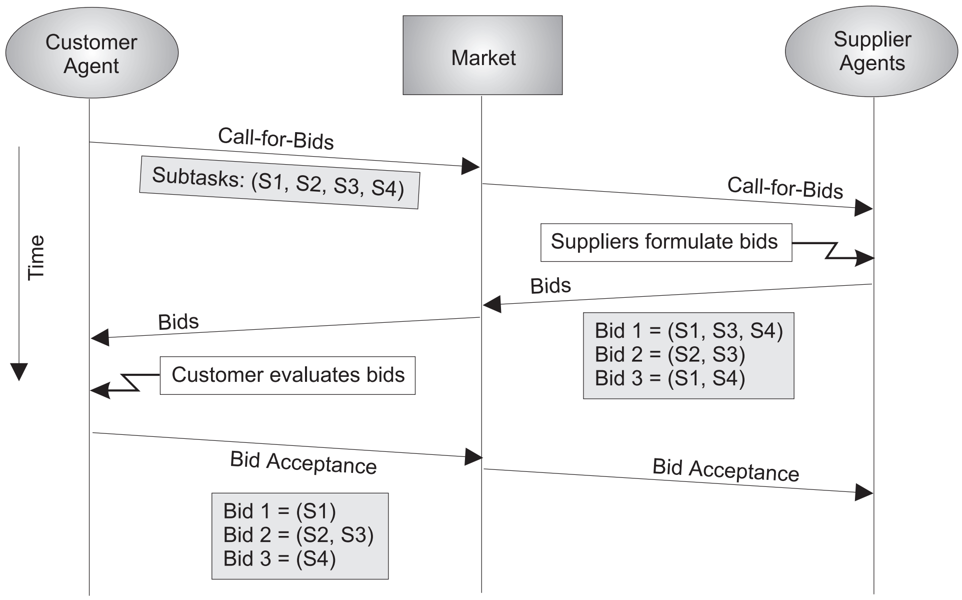

Fig. 5. A Typical contracting situation

We now present our bid selection algorithm. The goal is to find the best combination of bids and parts of bids (selecting only some of the subtasks from a bid, and ignoring the discount) to cover the entire set of subtasks specified by the call-for-bids.

The algorithm has two phases. First, we build an initial solution from the best individual subtask prices. If there are no bids for one or more of the subtasks, no initial solution can be constructed and the algorithm terminates. If a solution exists, we try to improve the initial solution by applying discounts from the various bids. Because each bid represents a single discount, we conduct our search by bid, not by subtask.

Each solution is represented by a node in a list of feasible solutions. We start the list by creating an initial solution, storing it in what we call the origin node, and placing the origin node in the feasible solution list.

Each node, which represents a solution to the problem, includes a list of subtasks, the price of each subtask, which bid is covering each subtask, whether each subtask is part of a discount (false for all subtasks in the origin node), and the total discount amount (zero in the origin node).

In the algorithm, we use the notation $node.bidID[i]$ to indicate the bid identifier of subtask i in the node $node$, $node.price[i]$ to indicate the price of subtask i, $node.discount?[i]$ to indicate if subtask i is part of a discount. $node.TotalDiscount$ indicates the total discount, $node.DiscountedPrice$ the discounted price, and $node.TotalPrice$ the total price of the solution. We will use a similar notation to indicate the components of a bid.

```
/* initialize origin node */
create origin node;
origin.TotalDiscount ← 0;
for each subtask ∈ SetofTasks do
    origin.bidID[subtask] ← unassigned
    origin.price[subtask] ← ∞
    origin.discount?[subtask] ← false

/* construct an initial solution (if one exists) */
for each bid ∈ SetofBids do
    for each subtask covered in bid do
        if      origin.bidID[subtask] = unassigned
                or bid.price[subtask] < origin.price[subtask]
        then    origin.price[subtask] ← bid.price[subtask]
                origin.bidID[subtask] ← bid.bidID[subtask]
solution? ← true
for each subtask ∈ SetofTasks do
    if origin.bidID[subtask] = unassigned then solution? ← false
if solution? = false then exit /* no solution exists */
add origin to SolutionList /* a solution exists */

/* improve the initial solution by applying one or more discounts */
for each bid ∈ SetofBids do
    for each node in SolutionList do
        discounted? ← false
        for each subtask covered in bid do
            if node.discount?[subtask] = true then discounted? ← true
        if      discounted? = false
                /* there is no subtask overlap for the discounts */
        then    create a new node current
                for each subtask in bid do
                    current.price[subtask] ← bid.price[subtask]
                    current.bidID[subtask] ← bid.bidID[subtask]
                    current.discount?[subtask] ← true
                current.TotalDiscount ← node.TotalDiscount + bid.discount
                current.TotalPrice ← Σ_{subtask∈SetofTasks} current.price[subtask]
                current.DiscountedPrice ←
                    current.TotalPrice − current.TotalDiscount
                if      current.DiscountedPrice < node.DiscountedPrice
                then    add current to TemporaryList
                else    discard it
    add the nodes from TemporaryList to SolutionList
    sort SolutionList in decreasing order by DiscountedPrice
the first node in SolutionList is the best solution
```

Let us now consider a detailed example of this procedure. In this example, we consider a call-for-bids on four subtasks. Suppose that, in response to the call-for-bids, three bids are received by the customer agent:

1. Bid 1 covers subtasks 1, 3 and 4 for 130 units with subtask 1 at 50 units, subtask 3 at 50 units and subtask 4 at 45 units (15 units discount).
2. Bid 2 covers subtasks 2 and 3 for 95 units with subtask 2 at 60 units and subtask 3 at 70 units (35 units discount).
3. Bid 3 covers subtasks 1 and 4 for 95 units with subtask 1 at 75 units and subtask 4 at 40 units (20 units discount).

The origin node is formed by taking the smallest individual price for each subtask, thus:

Origin		Parent Node: None	
subtask	bidID	price	discount?
1	1	50	false
2	2	60	false
3	1	50	false
4	3	40	false
total price:		200	
total discount:		0	
discounted price:		200	

We now try to form a child node for each node in the list using the Bid 1 discount. Since there is only one node in the list, and none of its subtasks are marked as discounted, we make a child node:

Node 1		Parent Node: Origin	
subtask	bidID	price	discount?
1	1	50	true
2	2	60	false
3	1	50	true
4	1	45	true
total price:		205	
total discount:		15	
discounted price:		190	

Since the discounted price is indeed less than the discounted price of its parent, we add this node to the list. We now try to create children using the Bid 2 discount. From the Origin Node we can make a child:

Node 2		Parent Node: Origin	
subtask	bidID	price	discount?
1	1	50	false
2	2	60	true
3	2	70	true
4	3	40	false
total price:		220	
total discount:		35	
discounted price:		185	

Since the discounted price is less than the discounted price of its parent, we add this node to the list. We cannot, however, make a child node from Node 1 (because there is a discount overlap on subtask 3).

We now move on to Bid 3. We can make a node from the Origin Node:

Node 3		Parent Node: Origin	
subtask	bidID	price	discount?
1	3	75	true
2	2	60	false
3	1	50	false
4	3	40	true
total price:		225	
total discount:		20	
discounted price:		205	

This node is not added to the list. Its discounted price is actually above the price of its parent (in this case the origin node).

We cannot make a child from Node 1 using Bid 3 because of the overlap on subtasks 1 and 4. We can, however, make a child of Node 2:

Node 4		Parent Node: Node 2	
subtask	bidID	price	discount?
1	3	75	true
2	2	60	true
3	2	70	true
4	3	40	true
total price:		245	
total discount:		55	
discounted price:		190	

This node is not added to the list, because though it is cheaper than the Origin Node, it is not cheaper than its parent node (Node 2).

There are now a total of three nodes in the list, and the cheapest price can be found in Node 2. Though that node contains higher subtask prices than the origin node, it contains enough discount to make it the least expensive combination.

The number of nodes created by this algorithm is highly dependent on the interaction between the number of bids, subtasks, price variation, and discount. We shall examine the results of some of these interactions in the next section.

Our algorithm conducts a systematic search on a finite space, so the algorithm is complete. It finds the optimal solution because it creates all non-conflicting discount combinations. Combinations which are not considered as solutions are rejected because they increase the total price. Since the algorithm starts with a solution and only combinations that decrease the price are considered, the algorithm has an anytime behavior. The algorithm can be terminated any time and will return the best solution found so far. Given additional time, it will produce a better solution, if one is available.

5 Experimental Evaluation

In order to observe the behavior of this algorithm under different circumstances, we constructed a set of experiments using the following parameters:

– The number of subtasks in the call-for-bids. We tried 10, 20 or 30 subtasks.
– The number of bids (suppliers). We tried 10, 20, or 30 suppliers.
– The mean percentage of subtasks that suppliers will include in their bids. This percentage was fixed at 30% for one set of experiments, and was varied randomly within the 10 to 60% range, for another set of experiments.
– The price range that suppliers can bid for each subtask. We tried allowing the price to vary widely (10-100) or narrowly (80-100).
– The percentage discount that suppliers will offer in their bids. This was picked with a uniform distribution within the range 0-40%.

All of the subtasks were considered to be of equal importance and were bid by the suppliers up to a price of 100 units each. Subtask ordering and other temporal considerations were ignored. For each experiment, ten different bid sets were produced with the same parameters, and the number of nodes examined to complete the search was computed.

Figures 6 and 7 illustrate the results for these experiments. Figure 6 shows the results for two sets of experiments, one in which the percentage of subtasks per bid was fixed at 30% and another with the percentage varying in the range of 11-60%. In both cases, the bid prices varied from 10 to 100 units. Figure 7 shows the results of another two sets of experiments using the same subtask percentage parameters, but using a bid price range of 80-100 units.

Comparing Figures 6 and 7, we can see that when pricing is allowed to fluctuate widely, the number of nodes searched decreases as the number of bids increases. When prices are constrained in their range, however, the number of nodes increases as the number of bids increases. This is due to the fact that, in the unconstrained scenario, there is an increased chance of bids being overpriced with

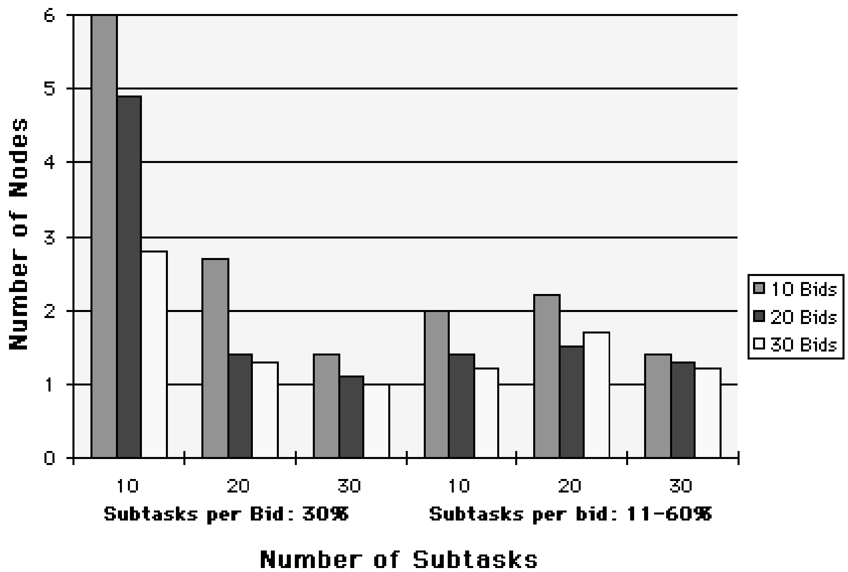

Fig. 6. Price varying from 10 to 100 unit

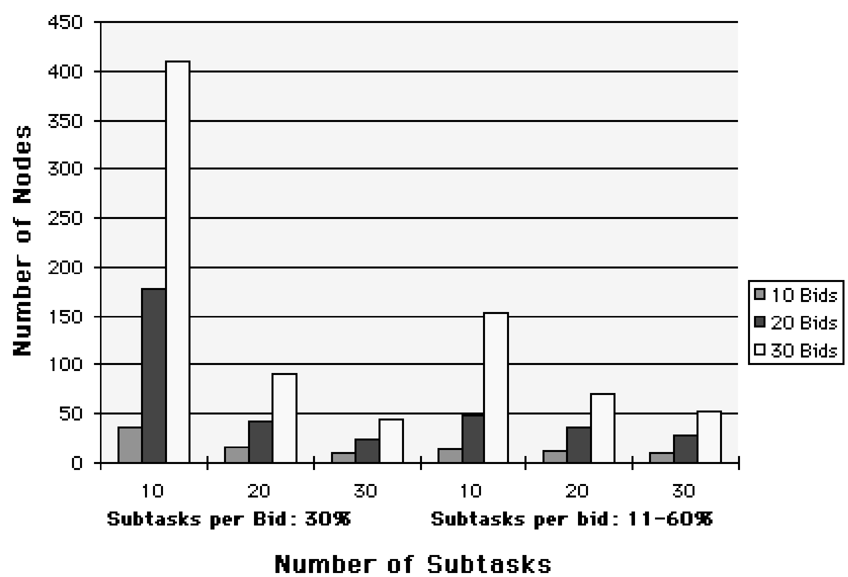

Fig. 7. Price varying from 80 to 100 unit

respect to the lowest price, even when considering their discount. This, in turn, results in an increase in the number of nodes discarded. In a typical contracting situation we should expect the price range not to have a large variance. Therefore it would be desirable for the customer agent to receive fewer bids, as illustrated in Figure 7.

When the subtask percentage (the percentage of subtasks that can appear in a bid) is allowed to vary up to sixty percent, some of the bid sets have a large number of subtasks, which causes the number of nodes searched to decrease as the number of subtasks increases. In general, the larger is the percentage of subtasks in each bid, the better the algorithm performs. At one extreme, if no bid contains multiple subtasks only one node is expanded. At the other extreme, if each bid includes all the subtasks, the algorithm is linear in the number of bids.

In Figure 8 we compared the performance of this algorithm with a standard A^* algorithm, using a minimum cost heuristic. As the figure shows, the number of nodes expanded by A^* grows very rapidly. A comparison with other branch and bound algorithms [10] is planned for the near future.

No. of Subtasks	No. of Bids	A^*	Anytime Algorithm
4	4	137	2.6
4	6	350	2.3
4	8	695	1.8
6	4	659	2.7
6	6	3682	2.1
6	8	7367	1.9
7	4	4830	2.3
7	6	22104	1.5

Fig. 8. Number of nodes expanded by A^* and by the anytime algorithm for a variety of problems. For all the experiments the price range is between 10 and 100 units, the percentage of subtasks each suppliers includes in the bids is between 30% and 80%. The table shows the average number of nodes expanded in 10 runs for each experiment.

From these results we can see that the interesting parameters to explore should be when the percentage of subtasks that can appear in a bid is small and both prices and discounts are kept in a reasonable range. Under these conditions, the space searched can become very large with larger numbers of bids and subtasks. In order to use the anytime property of this algorithm, it may become useful to sort the bids (and thus guide the search space) by the percentage discount given. When the algorithm is interrupted, it will have already tried to apply the better discounts, and so should produce a cheaper solution than looking at the bids in a random order.

In a further experiment, we limited the prices and discounts to a reasonable range. The prices were kept between 80% and 100% of the highest price, and discounts were allowed only up to 30% of the total price of a bid. Further, the subtask percentage was kept at 10%. We then looked at the effect of varying the number of bidders from 20 to 45 and the number of subtasks from 20 to 45.

The following table shows the results of our experiments. Each cell shows the mean number of nodes expanded for trials in which a feasible solution existed. Ten trials were attempted for each cell. Assuming that decisions to bid on individual subtasks are independent events, the probability that all subtasks will be bid on by at least one bidder is $\mathcal{P} = (1 - (1 - p)^m)^n$, where m is the number of bidders, n is the number of subtasks, and p is the probability that a bidder will bid on a subtask. It should be noted that the variance in these numbers is rather high; typically, $\sigma > 0.6\overline{X}$.

Bidders / Subtasks	20	25	30	35	40	45
20	364	1352	8125	27591	72465	201827
25	2190	6827	15384	34064	66510	88380
30		1366	3271	19244	45595	85348
35		2767	9613	21659	31409	55318
40			4088	7493	21257	34133
45			4445	7167	16136	32795

Fig. 9. Number of nodes expanded by the anytime algorithm for a price range between 80 and 100 units, the probability of a subtask being included in a bid is 10%. Empty cells had no instances of full subtask coverage in 10 runs.

Under these conditions, it appears as though the number of nodes searched increases exponentially with the number of bidders when the number of subtasks is kept constant, approximately doubling with every five bidders added. The number of nodes searched decreases, however, as the number of subtasks increases for a constant number of bidders, which also increases the probability that some subtasks will be included in only one or a very small number of bids.

There are two ways this information could be used by a customer agent in the MAGNET system: before the Call For Bids is issued, and after bids are received:

– if a customer agent has a priori knowledge of the likely number of bidders and the bid density (expected number of subtasks per bid), then the structure of the Call For Bids could be manipulated to both increase the probability of achieving plan coverage, and to reduce the search effort. Such manipulation could be done by choosing plan expansions with more or fewer elements, or with different levels of hierarchical breakdown. The necessary a priori

knowledge could be gathered by the Market as contracting activity proceeds under its jurisdiction;
– after bids are received, a simple measure of the bid density and degree of overlap could be used to estimate the required search effort. If the predicted effort was greater than the available time, then a different search strategy, such as simulated annealing, might be chosen in order to achieve broad coverage of the search space, while sacrificing detailed examination.

6 Related Work

In recent years, a variety of architectures have been proposed for electronic commerce and multi-agent automated contracting [4,9,13,15,17,24,25].

In addition to the work on virtual market architectures, several protocols have been developed and proposed that support automated contracting and negotiation among multiple agents in such markets [11,18–20]. Automated contracting protocols generally assume direct agent-to-agent negotiation. For example, Smith [23] pioneered research in communication among cooperating distributed agents with the Contract Net protocol. The Contract Net has been extended by Sandholm and Lesser [19] to self-interested agents.

In these systems, agents communicate and negotiate directly with each other. On the other hand, in the MAGNET system [7], the proposed architecture and the associated protocol for automated contracting utilize an external and independent market infrastructure to reduce fraud and counterspeculation among self-interested agents. In contrast to Sandholm's protocol [20], MAGNET avoids the need for open-ended negotiation by means of bid break-downs and time-based decommitment penalties, as described more in detail in [6].

A primary motivation behind the design of our proposed protocol and market framework is to support automated contracting. This sort of problem is often found in public contracting and it is useful, in general, in multi-enterprise manufacturing.

Existing architectures are generally designed for the kind of commercial activity that involves buying and selling of physical or electronic goods over a distributed electronic environment such as the Internet. They do not explicitly support more complex interactions such as those in a contracting domain where customer agents formulate plans and use the negotiation process to gain commitment from multiple supplier agents for the execution of these plans.

To the extent that we require the existence of an external market mechanism as an intermediary, our proposed framework is similar to that of Wellman's market-oriented programming used in AuctionBot [26]. AuctionBot supports a variety of auction types each imposing a set of market rules on the agent interactions. Hence, the auctions, themselves, become the intermediaries. The entity that sets up the auction can specify certain parameters for the auctions. In contrast, our framework provides explicit market mechanisms which can not only specify and enforce auction parameters, but also support more complex interactions. Furthermore, these market mechanisms also enforce general market rules and "social laws", such as government regulations, by which all participants must abide. Rosenschein and Zlotkin [18] 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 Rosenschein and Zlotkin' study [18] the agents are homogeneous, and the assumption is that there are no side payments. In other words, the goal is to share the work, not to pay for work. Sandholm's agents [19,20,1] redistribute work among themselves by a contracting mechanism. Unlike Rosenschein and Zlotkin, Sandholm considers agreements involving explicit payments.

7 Conclusions and Future Work

In this paper we have presented an overview of the MAGNET automated contracting system, and preliminary results of our work in developing an anytime algorithm that can choose the best combination of bids in real time on a reasonably sized problem. Our proposed algorithm has been developed as part of the MAGNET contracting market framework [7]. It compares favorably with algorithms that build solutions (for example, a constructive A^* search of the subtask space).

Our experimental evaluation suggests that the algorithm searches very efficiently and expands a small number of nodes before producing the optimal solution. The algorithm can be interrupted at any time and will return the best solution found so far. Our results also affirm the common sense notion that there is a tradeoff between cost of computation and opportunity for optimization.

It has been observed that there is often a form of *phase transition* situation that separates easy from hard problems [5]. This observation has produced significant results in the context of propositional satisfiability (SAT) problems (see, for instance, [14,12]. It would be worthwhile to explore if specific heuristics adapt better to either of these extremes, and to study the effect of alternative pruning tactics on hard problems in the domain we have described here.

There are extensions to this algorithm that we are considering. First, we plan on including other factors in the cost of bids, such as the reliability of the supplier, or the desirability of the customer to deal with a specific supplier, Second, we plan on extending the algorithm to include time considerations in addition to price. The best bid could be the one that accomplishes the task at the most appropriate time for the customer, not the one that has the lowest price.

References

1. Martin R. Andersson and Tuomas W. Sandholm. Sequencing of contract types for anytime task reallocation. In *1998 Workshop on Agent Mediated Electronic Trading*, Minneapolis, MN, May 1998.
2. Mark Boddy. Anytime problem solving using dynamic programming. In *AAAI91*, pages 738–743, 1991.

3. Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *International Joint Conference on Artificial Intelligence*, pages 979–984, 1989.
4. Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proc. of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, April 1996.
5. P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 331–337, 1991.
6. 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.
7. John Collins, Ben Youngdahl, Scott Jamison, Bamshad Mobasher, and Maria Gini. A market architecture for multi-agent contracting. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 285–292, May 1998.
8. Robert Doorenbos, Oren Etzioni, and Daniel Weld. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the First International Conference on Autonomous Agents*, pages 39–48, 1997.
9. Joakim Eriksson, Niclas Finne, and Sverker Janson. SICS Marketspace – an agent-based market infrastructure. In *1998 Workshop on Agent Mediated Electronic Trading*, pages 33–48, Minneapolis, Minnesota, May 1998.
10. Freuder and Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
11. Robert H. Guttman and Pattie Maes. Agent-mediated integrative negotiation for retail electronic commerce. In *1998 Workshop on Agent Mediated Electronic Trading*, pages 77–90, Minneapolis, Minnesota, May 1998.
12. Larrosa and Meseguer. Phase transition in MAX-CSP. In *Proceedings of the European Conference on Artificial Intelligence*, pages 190–194, 1996.
13. S. McConnell, M. Merz, L. Maesano, and M. Witthaut. An open architecture for electronic commerce. Technical report, Object Management Group, Cambridge, MA, 1997.
14. D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 459–465, 1992.
15. Tracy Mullen and Michael P. Wellman. The auction manager: Market middleware for large-scale electronic commerce. In *1998 Workshop on Agent Mediated Electronic Trading*, pages 113–128, Minneapolis, Minnesota, May 1998.
16. Colin R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* John Wiley & Sons, New York, NY, 1993.
17. J. A. Rodriguez, Pablo Noriega, Carles 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.
18. Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter.* MIT Press, Cambridge, MA, 1994.
19. Tuomas Sandholm and Victor Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *1st International Conf. on Multiagent Systems*, pages 328–335, San Francisco, 1995.
20. Tuomas W. Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents.* PhD thesis, University of Massachusetts, 1996.

21. Tuomas W. Sandholm and Victor R. Lesser. Advantages of a leveled commitment contracting protocol. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 126–133, Portland, Oregon, July 1996. AAAI.
22. Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, CA, 1992. AAAI, AAAI Press.
23. R. G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Trans. on Computers*, 29(12):1104–1113, December 1980.
24. J. M. Tennenbaum, T. S. Chowdhry, and K. Hughes. eCo System: CommerceNet's architectural framework for internet commerce. Technical report, Object Management Group, Cambridge, MA, 1997.
25. Maksim Tsvetovatyy, Maria Gini, Bamshad Mobasher, and Z. Wieckowski. MAGMA: An agent-based virtual market for electronic commerce. *Journal of Applied Artificial Intelligence*, 11(6), 1997.
26. P.R. Wurman, M.P. Wellman, and W.E. Walsh. The Michigan Internet Auctionbot: A configurable auction server for human and software agents. In *Second Int'l Conf. on Autonomous Agents*, pages 301–308, May 1998.
27. 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.