An Efficient Anytime Algorithm For Multiple-Component Bid Selection in Automated Contracting

Erik Steinmetz, John Collins, Maria Gini, and Bamshad Mobasher

Department of Computer Science and Engineering
University of Minnesota

Abstract. 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. The job of the customer agent is to select the optimal bid combination. In our experiments 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

In recent years, a variety of architectures have been proposed for electronic commerce and multi-agent automated contracting [3, 7, 10, 13, 18, 19].

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 [14, 15, 16]. For example, Smith [17] pioneered research in communication among cooperating distributed agents with the Contract Net protocol. The Contract Net has been extended by Sandholm and Lesser [15] to self-interested agents.

In these systems, agents communicate and negotiate directly with each other. On the other hand, in the MAGNET (Multi AGent NEgotiation Testbed) system [6], 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.

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.

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.

The MAGNET contracting market framework incorporates a three step process with a customer agent issuing a call-for-bids, suppliers replying with bids, and the customer accepting the bids it chooses with bid-accept messages. The bid is a commitment by the supplier to do the work listed in the bid, should the customer accept it.

In contrast to Sandholm's protocol [16], 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 [5].

Regardless of the specific protocol used for agent negotiation, once the customer receives the bids from supplier agents, it must evaluate the bids based on cost and/or time constraints, and select the optimal set of bids (or parts thereof) which can satisfy its goals.

To avoid open ended negotiation we require the suppliers to itemize prices of individual subtasks. Since the customer knows what the supplier will charge for each individual subtask, if the customer decides to accept only a subset of the original bid there is no need for additional negotiation.

Considering that customer and suppliers are software agents residing on different machines and communicating over the Internet, any reduction in communication has the advantage of increasing the speed and robustness of the overall system.

In this paper, we focus on the bid selection methods used by a customer agent to evaluate supplier bids and formulate a contract. In particular, we propose an *anytime* algorithm [2] which allows the customer agent to obtain the optimal bid combination based on a cost function. What makes the problem difficult is the presence of discounts in bids for combinations of subtasks.

We have chosen a local improvement search over a constructive search for three reasons. First, there is a straightforward mechanism for constructing a baseline feasible solution.

Second, 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 search as an anytime search. In [1], Boddy has further characterized the requirements for anytime problem solving using performance profiles.

Third, 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 [12].

The rest of this paper is organized as follows. In Section 2 we provide the details of our anytime algorithm and show how it operates on a simple example. In Section 3 we describe the results of our experimental evaluation which explore the behavior of the algorithm based on the interactions of factors such as bid prices, number of bids, and number of subtasks. We then compare this algorithm to the standard admissible A^* algorithm, using a minimum cost heuristic. Finally, in Section 4, we conclude by discussing some of the areas that remain to be explored in our future work.

2 Description of the Algorithm

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 (as in MAGNET [6]) 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 1, 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.

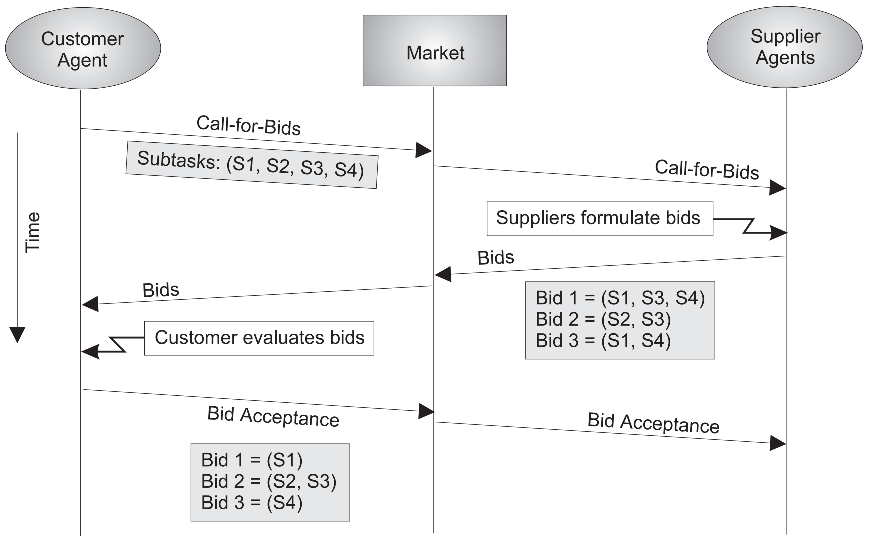

Fig. 1. A Typical contracting situation

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;
$origin.TotalDiscount \leftarrow 0$;
for each $subtask \in SetofTasks$ do
    $origin.bidID[subtask] \leftarrow unassigned$
    $origin.price[subtask] \leftarrow \infty$
    $origin.discount?[subtask] \leftarrow false$

/* construct an initial solution (if one exists) */

for each $bid \in SetofBids$ do
    for each $subtask$ covered in $bid$ do
        if      $origin.bidID[subtask] = unassigned$
                or $bid.price[subtask] < origin.price[subtask]$
        then $origin.price[subtask] \leftarrow bid.price[subtask]$
                $origin.bidID[subtask] \leftarrow bid.bidID$[subtask]
$solution? \leftarrow true$
for each $subtask \in SetofTasks$ do
    if $origin.bidID[subtask] = unassigned$ then $solution? \leftarrow false$
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 $bid \in SetofBids$ do
    for each $node$ in $SolutionList$ do
        $discounted? \leftarrow false$
        for each $subtask$ covered in $bid$ do
            if $node.discount?[subtask] = true$ then $discounted? \leftarrow true$
        if   $discounted? = false$
             /* there is no subtask overlap for the discounts */
        then create a new node $current$
             for each $subtask$ in $bid$ do
                 $current.price[subtask] \leftarrow bid.price[subtask]$
                 $current.bidID[subtask] \leftarrow bid.bidID[subtask]$
                 $current.discount?[subtask] \leftarrow true$
             $current.TotalDiscount \leftarrow node.TotalDiscount + bid.discount$
             $current.TotalPrice \leftarrow \sum_{subtask \in SetofTasks} current.price[subtask]$
             $current.DiscountedPrice \leftarrow$
                 $current.TotalPrice - current.TotalDiscount$
             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.

3 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 2 and 3 illustrate the results for these experiments. Figure 2 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 3 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 2 and 3, 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 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 3.

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 4 we compared the performance of this algorithm with a standard A^* algorithm, using a minimum cost heuristic. As the figure shows, the number

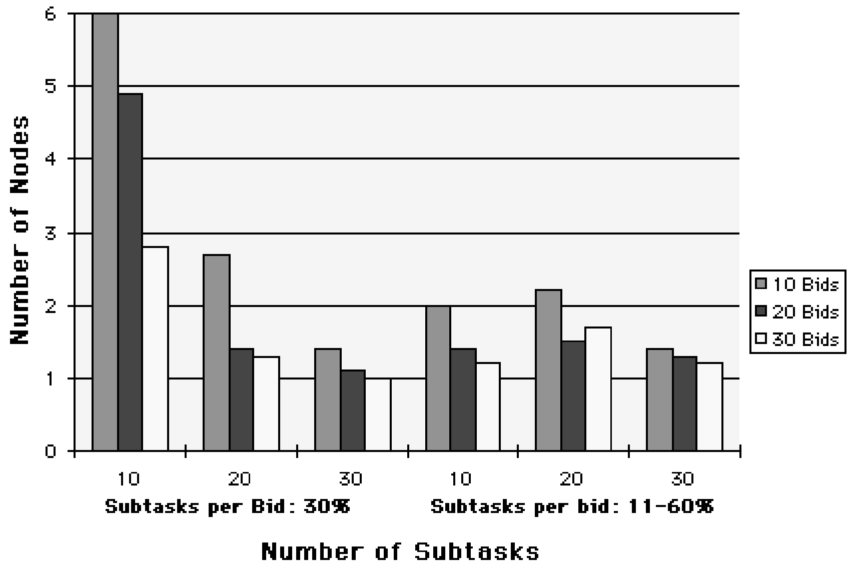

Fig. 2. Price varying from 10 to 100 unit

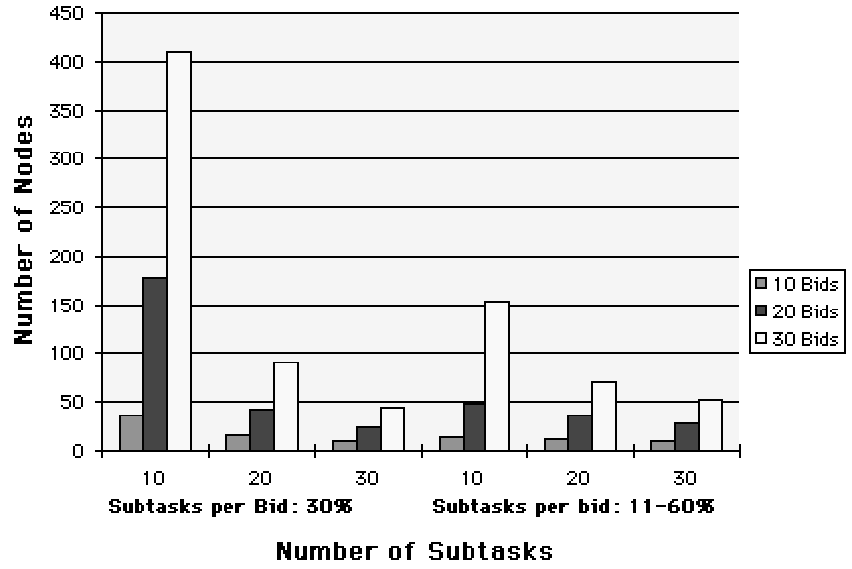

Fig. 3. Price varying from 80 to 100 unit

of nodes expanded by A^* grows very rapidly. A comparison with other branch and bound algorithms [8] 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. 4. 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.

All of the experiments implemented here were coded in Java 1.02, compiled using Code Warrior(tm), and run on a BeBox. On average about one hundred nodes per second were added to the solution lists, so our algorithm produced solutions in most of the experiments within one to two seconds.

From these results we can see that the interesting parameters to explore should be when the subtask bid percentage 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.

4 Conclusions and Future Work

In this paper we have presented preliminary results of our work in developing an anytime algorithm that can operate in an electronic marketplace of agents, and 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 [6]. 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 the search space 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.

It has been observed that there is often a form of *phase transition* situation that separates easy from hard problems [4]. This observation has produced significant results in the context of propositional satisfiability (SAT) problems (see, for instance, [11, 9]. 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. M. Boddy. Anytime problem solving using dynamic programming. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 738–743, 1991.
2. M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh Joint Conference on Artificial Intelligence*, 1989.
3. 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.
4. 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.
5. J. Collins, S. Jamison, M. Gini, and B. Mobasher. Temporal strategies in a multi-agent contracting protocol. In *AAAI-97 Workshop on AI in Electronic Commerce*, July 1997.
6. J. Collins, B. Youngdahl, S. Jamison, B. Mobasher, and M. Gini. A market architecture for multi-agent contracting. In *Proceedings of the Second Internatoinal Conference on Intelligent Agents*, May 1998.
7. J. Eriksson and N. Finne. Marketspace: an open agent-based market infrastructure. Master's thesis, Uppsala University, Computing Science Department, 1997.
8. Freuder and Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
9. Larrosa and Meseguer. Phase transition in MAX-CSP. In *Proceedings of the European Conference on Artificial Intelligence*, pages 190–194, 1996.
10. S. McConnell, M. Merz, L. Maesano, and M. Witthaut. An open architecture for electronic commerce. Technical report, Object Management Group, Cambridge, MA, 1997.
11. 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.

12. Colin R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* John Wiley & Sons, New York, NY, 1993.
13. J. A. Rodriguez, F. 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.
14. J. S. Rosenschein and G. Zlotkin. *Rules of Encounter.* MIT Press, Cambridge, MA, 1994.
15. T. Sandholm and V. Lesser. Issues in automated negotiation and electronic commerce: Extending the Contract Net framework. In *1st International Conf. on Multiagent Systems (ICMAS-95)*, pages 328–335, San Francisco, 1995.
16. T. W. Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents.* PhD thesis, University of Massachusetts, 1996.
17. R. G. Smith. The Contract Net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, December 1980.
18. 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.
19. M. Tsvetovatyy, M. Gini, B. Mobasher, and Z. Wieckowski. MAGMA: An agent-based virtual market for electronic commerce. *Journal of Applied Artificial Intelligence*, (6), 1997.