In *AAMAS-04 Workshop on Agent Mediated Electronic Commerce* (AMEC 04),
New York, New York, USA, July 2004.

# Bidding for Customer Orders in TAC SCM

David Pardoe and Peter Stone

The University of Texas at Austin, Austin TX 78712, USA
{dpardoe, pstone}@cs.utexas.edu

**Abstract.** Supply chains are a current, challenging problem for agent-based electronic commerce. Motivated by the Trading Agent Competition Supply Chain Management (TAC SCM) scenario, we consider an individual supply chain agent as having three major subtasks: acquiring supplies, selling products, and managing its local manufacturing process. In this paper, we focus on the sales subtask. In particular, we consider the problem of finding the set of bids to customers in simultaneous reverse auctions that maximizes the agent's expected profit. The key technical challenges we address are i) predicting the probability that a customer will accept a particular bid price, and ii) searching for the most profitable set of bids. We first compare several machine learning approaches to estimating the probability of bid acceptance. We then present a heuristic approach to searching for the optimal set of bids. Finally, we perform experiments in which we apply our learning method and bidding method during actual gameplay to measure the impact on agent performance.

## 1 Introduction

Supply chains are a current, challenging problem for agent-based electronic commerce. One problem commonly faced by agents acting in supply chains is that of negotiating with customers in order to sell goods. Such negotiations are often handled through reverse auctions in which sellers submit sealed bids in response to requests for quotes (RFQs) from customers. This situation becomes particularly difficult when sellers must bid in multiple auctions simultaneously, because an agent cannot await the outcome of one auction before bidding in another. When deciding which auctions to bid in and what bids to place, an agent with limited resources must be able to judge and balance the competing risks of not winning enough auctions and of winning too many. In the former case, it is unable to fully utilize its resources towards profitability; in the latter, it will be unable to meet its obligations to customers.

The Trading Agent Competition Supply Chain Management (TAC SCM) scenario [1] provides a perfect testbed for the study of this problem. In TAC SCM, agents competing as computer manufacturers must handle three basic subtasks: acquiring components, managing a local manufacturing process, and selling assembled computers to customers. Agents receive incomplete information about the state of the game and have a limited amount of time in which to make decisions, resulting in a challenging competition. The problem studied in this paper is motivated by our work on TacTex [2], the third place entry from the first TAC SCM competition. From our experience, we have identified the sales subtask as the most crucial aspect of the TAC SCM scenario.

In this paper, we focus on the problem of determining the optimal set of bids for an agent to make in response to RFQs for computers received from customers. The key technical challenges we address are i) predicting the probability that a customer will accept a particular bid price, and ii) searching for the most profitable set of bids.

The remainder of this paper is organized as follows. In Sect. 2 we give a brief summary of the TAC SCM scenario and provide information on related work. We give a complete description of the problem we are solving in Sect. 3. In Sect. 4 we present a comparison of several machine learning approaches to estimating the probability of bid acceptance. We describe a heuristic approach to finding an optimal set of bids in Sect. 5. In Sect. 6 we measure the impact of learning on agent performance by performing controlled experiments involving actual TAC SCM games. Sect. 7 proposes directions for future work and concludes.

## 2 Background

In this section, we give a brief summary of the TAC SCM scenario, emphasizing the parts that are most relevant to the sales subtask, and provide information on related work.

### 2.1 The TAC SCM Game

In a TAC SCM game [3], six agents act as computer manufacturers in a simulated economy that is managed by a game server. The length of a game is 220 simulated days, with each day lasting 15 seconds of real time. At the beginning of each day, agents receive messages from the game server with information concerning the state of the game, such as the customer RFQs for that day. Agents have until the end of the day (i.e. $< 15s$) to send messages to the server indicating their actions for that day, such as bids on RFQs. The game can be divided into three parts: production and delivery, component supply, and computer demand.

In this paper, we focus on the computer demand, or sales, aspect of the TAC scenario. Customers wishing to buy computers send all six agents identical RFQs consisting of:

- the type of computer desired (1 of 16);
- the quantity of computer desired (1–20);
- the due date (3-12 days in the future);
- a reserve price indicating the maximum amount the customer will pay; and
- a penalty that must be paid for each day the delivery is late. Orders are canceled on the fifth late day.

Reserve prices range from 75% to 125% of the base price of the requested computer type, multiplied by the quantity, and penalties range from 5% to 15% of the reserve price. The base price of a computer is equal to the sum of the base prices of its parts [3]. Agents respond to the RFQs by making offers to sell at a certain price, with the agent offering the lowest bid on each RFQ winning the order. Agents are unable to see the prices offered by other agents or even the winning prices, but they do receive a report each day indicating the highest and lowest price at which each type of computer sold on the previous day.

The number of RFQs that come from customers depends on the level of customer demand, represented by a parameter $D$. The actual number of RFQs each day is drawn from a Poisson distribution with $D$ as its mean. Fluctuation in demand is modeled by multiplying $D$ by an amount representing the current trend each day. This trend follows a random walk, and $D$ is bounded between 80 and 320, with its initial value chosen uniformly randomly from this range.

### 2.2 Related Work

The problem of predicting the probability of winning an auction with a particular sealed bid is commonly approached through statistical methods such as those surveyed in [4]. Such methods often require extensive historical information about competitors' past bids and assume a static environment. In TAC SCM, probabilities vary considerably throughout the game, and almost no information is available about competitors' bids while the game is running. A machine learning approach similar to that used in this paper is developed by [5], which uses a naive Bayes classifier to predict the probability of a bid winning based on the bid price, features of the RFQ , and available information about other bidders.

A solution to the TAC SCM bidding problem similar to the one used in this paper is presented in [6], which uses linear regression on recent bidding results to form predictions of bid acceptance and then uses stochastic programming to determine optimal bids. Additional approaches are described in [7] and [8].

## 3 Problem Specification

We now specify the problem we are addressing in this paper. We consider the problem of an agent participating in a TAC SCM game that must decide what bids to place on the RFQs it has received from customers on a given day. The inputs to the agent's decision process are the following:

- The set of customer RFQs;
- The agent's available resources (components and assembled computers in inventory along with the future production cycles); and
- Information about past auctions (the agent's knowledge of its own bids and the reported highest and lowest prices at which each type of computer sold)

Because there are many more computers requested each day than one agent can produce, the goal of an agent is not to win every auction, but to find the set of bids that maximizes the agent's expected profit without committing the agent to produce more computers than it possibly can. (Viewing TAC as a game, an agent's goal should be to maximize its profit relative to the profits of competing agents, but due to the difficulty of determining the effect an agent's bids will have on other agents, we will assume our agent is only concerned with its own profit. In a real supply chain, this profit maximization would be the true goal.) A simple approach to this problem is to predict the highest price at which each auction could be won and to bid this price on several of the more profitable auctions, expecting to win each one. A more sophisticated approach involves considering the possibility of placing high bids on many auctions in hopes of winning some fraction of them.

This second approach is the one used by TacTex, our agent in the first TAC SCM competition, and is the approach that is considered in this paper. An agent implementing this approach has two requirements: the ability to form estimates of the probability of winning an auction as a function of the bid price, and a means of using these estimates to find the set of bids that maximizes the agent's expected profit. We consider these two agent components separately in the next two sections. In Sect. 4, we experiment with different machine learning approaches to predicting the probability of bid acceptance, and in Sect. 5, we present a heuristic approach to the problem of bid selection.

## 4 Learning Auction-Winning Probabilities

Predicting the probability of winning an auction in TAC SCM is a challenging problem for three main reasons: (i) agents receive very limited information on auction results, (ii) no two auctions are the same due to the differing attributes of each RFQ, and (iii) winning prices can fluctuate rapidly due to changing game conditions. As a result, an approach based on analyzing past auction results from the current game is unlikely to yield accurate predictions. We therefore turn to machine learning methods using training data from many past games.

The problem we are trying to solve can be viewed as a multiple regression problem. This could be solved by using a regression learning algorithm to learn the probability of winning an auction as a function of factors including the bid price. We instead follow a modified approach used by [9] to solve a similar conditional density estimation problem from a different TAC scenario. This approach involves dividing the price range into several bins and estimating the probability of winning the auction at each bin endpoint. A post-processing step converts the learned set of probabilities to a probability density function by interpolating between bin endpoints and enforcing a monotonicity constraint that ensures that probabilities decrease as prices increase. In this method, a separate predictor is trained for each endpoint to predict the probability of winning at that point. The concept to be learned by each predictor is therefore simpler than the concept that would be learned if we used a single predictor for all prices. We leave an empirical comparison with the latter approach for future work.

In this section we focus on the task of training these individual predictors. We describe the format of the training data, compare the effectiveness of several learning algorithms, and then look at the impact that the choice of training data has on the predictions. It is important to note that training is done off-line, so the game's time constraints are not a factor.

### 4.1 Training Data Format

The data for our experiments is taken from the results of the semifinal and final rounds of the first TAC SCM competition held in August 2003. Winning bids for customer RFQs can be obtained from game logs made available immediately after each game terminates. Several hundred thousand RFQs were issued over the course of the games, providing ample data for training and testing.

A training instance is created for each RFQ. The 23 attributes included in each instance reflect the details of the RFQ it represents, along with the information available to agents at the time about the level of demand in the game and the recent prices for which the requested type of computer has been sold. Each instance contains the current date; the quantity, penalty, due date, and reserve price for the RFQ; and the highest and lowest prices at which the requested computer type was sold over the past five days. The additional attributes provided about customer demand give a picture of how the daily number of RFQs has varied over the course of the game. All monetary values are expressed as a fraction of the computer's base price.

A separate predictor is trained for each price point $x$ at which we want to predict the probability of winning an auction, where $x$ is expressed as a fraction of a computer's base price. For a given value of $x$, each auction is labeled with a 1 if the winning bid was greater than $x$ and with a 0 otherwise. Instances representing RFQs receiving no bids are labeled with a 1 if $x$ is less than or equal to the reserve price.[1]

### 4.2 Algorithm Comparison

We first performed an experiment comparing the effectiveness of using several different regression learning algorithms to train predictors: neural networks (with a single hidden layer and using backpropagation), M5 regression trees, M5 regression trees boosted with additive regression (which successively fits a new base learner to the residuals left from the previous step), decision stumps (single-level decision trees) boosted with additive regression, J48 decision trees, J48 decision trees boosted with AdaBoost, and BoosTexter. BoosTexter [10] is a boosting method that was originally designed for text classification and is the algorithm used in [9]. It uses decision stumps as the base learner and a form of AdaBoost to weight each training instance between rounds of training, outputting a weighted sum of the learned decision stumps. Other algorithms we considered were support vector machines, naive Bayes, and k-nearest neighbors, but these did poorly in initial testing. For all algorithms other than BoosTexter, we used the implementations provided in the WEKA machine learning package [11], using default parameters. Informal attempts at tuning these parameters did not appear to significantly affect performance.

For comparison, we also include the results obtained from using a simple heuristic predictor that gives reasonably good results. For each of the past five days, the predictor forms a uniform density function on the interval between the highest and lowest prices reported for the requested computer type. A weighted sum of these density functions, with those from more recent days receiving more weight, is then used as a probability density function from which estimates of bid acceptance are taken.

In our experiment we evaluated each of the learning algorithms on a data set taken from the final round of the competition. We used cross-validation, meaning

[1] Note that this formulation represents the standpoint of a seventh agent wanting to know the probability that none of the other six agents would place a bid below $x$.

that training and test data came from the same games. While the true value of a probability prediction is the utility gained from using it, determining this in the context of a TAC SCM agent is not feasible, and so we instead use root mean squared error between the predicted probabilities and actual outcomes as the measure of comparison. We ran separate tests predicting the probability of winning an auction at several different values of $x$. The results of three 10-fold cross-validations with $x = 0.7$ are presented in Fig. 1 and are similar to results for other values of $x$.

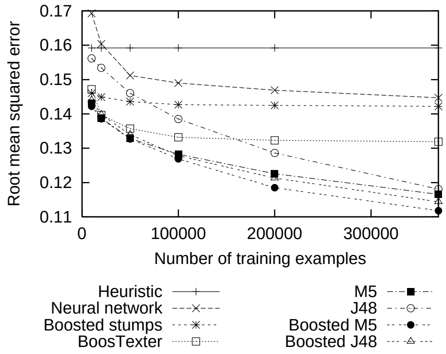


**Fig. 1.** Results for $x = 0.7$

With a large number of training instances, the tree-based methods clearly had the best performance, followed by BoosTexter. The errors of the non-tree-based methods level off after a limited number of training instances, while the errors of the tree-based methods continue to decrease until the point at which all available training data is used. For training sets of size 200,000 and 370,000, the difference observed between each pair of algorithms in Fig. 1 is statistically significant at the 95% confidence level.

### 4.3 Choice of Training Data

In the previous experiment, the training data and test data were taken from the same set of games, with the same agents participating in each game. This raises the possibility that the algorithms learned concepts that pertained to specific games and set of agents but were not applicable in general. This is also unrealistic in the TAC setting, as an agent could not have predictors trained on data from the game it is currently participating in. In practice, it is important to know whether a predictor trained for one set of agents will be reliable in games with a different set of agents. Our next experiment addresses these issues.

We consider the case of an agent participating in the final round of the competition. The agent would want to train predictors on the data most relevant to the situation. At the beginning of the final round, the most relevant data would likely come from the results of the semifinal round, which contained two brackets of six agents each. As a result, this data would reflect on both the agents in the final round and the agents defeated in the semifinal round. After the finals begin, the agent would be able to analyze the results of completed games from the finals and would have the option of retraining its predictors with this new data, either by itself or in combination with the data from the semifinals.

We performed an experiment comparing the results of training with these choices of training data. First, we divided the games from the final round into two halves, labeled finals1 and finals2. We then used finals2 as the test data for predictors trained on data from different sources: the semifinals, finals1, the semifinals combined with finals1, and finals2 (using cross validation). The results for M5 trees and BoosTexter, the top two performing algorithms, are shown in Figures 2 and 3. Again, $x = 0.7$. The learning curves are labeled with the source of data used for training.

When the predictors were trained on data other than finals2, the performance gap between M5 trees and BoosTexter disappeared, and the performance of the other tree-based methods, even boosted M5 trees, fell behind. The errors of the tree-based methods no longer continued to decrease as more training instances were used, and sometimes the error increased, as observed in Fig. 2 when data from the semifinals was used for training. This suggests that the strong performance of the tree-based methods in the first experiment was largely due to their ability to learn game-specific factors that do not generalize well. While BoosTexter appears to achieve somewhat lower errors than M5 trees in this experiment, further testing on different game scenarios would need to be done to determine whether this is the case in general.

As we expected, the predictors trained on data from finals2 outperformed the predictors trained on data from different games. Still, the performances of the latter were better than that of the heuristic. The predictors trained on finals1 performed better than those trained on the semifinals, confirming that more relevant training data produces better results. Somewhat surprisingly, the predictors trained on the combination of finals1 and the semifinals performed better than the predictors trained on finals1 only. It may be that a predictor trained on data from a variety of sources will generalize the best to a new situation, even if some of the training data is less relevant for the new situation.

The results of these experiments suggest that with the right choice of learning algorithm and training data, we can learn the probability of winning an auction reasonably well. However, to measure the value of our predictions, we need to use them as the input to a method of selecting bids in actual TAC games. We present such a method in the next section, and then experimentally evaluate its performance in Sect. 6.

## 5 Bid Selection

We now consider the problem of bid selection. Recall that each day, customers send roughly 80-320 RFQs to all agents simultaneously, with each RFQ request-

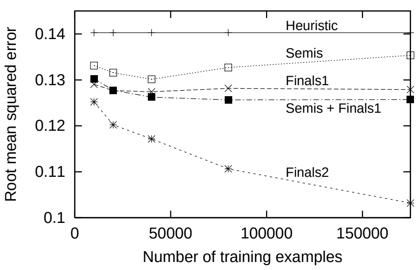


**Fig. 2.** M5 Trees

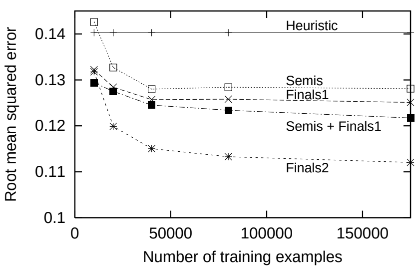


**Fig. 3.** BoosTexter

ing a specific type and quantity of computer by a certain date. For each RFQ, the agent that bids lowest wins the order. In this section, we cast the bidding problem as an optimization problem and describe a heuristic approach to finding the optimal set of bids to offer in response to a single day's customer RFQs.

### 5.1 Problem Formulation

An agent's goal in selecting bids should be to maximize its total expected future profit. This value depends on the unknown strategies of competing agents, however, and the exact computation of this value would likely be intractable even if these strategies were known. As a result, we present a somewhat myopic agent that aims to maximize its profit only on computers due over the next 12 days (the range of due dates for computers requested on a given day) and that makes some simplifying assumptions. These assumptions are:

1. All computers delivered over the next 12 days will come from computers and components that are already in inventory or expected to be delivered during that period.
2. After the next 12 days, the average number of each computer type ordered from the agent per day will remain the same as the average over the past few days.
3. For the rest of the game, the agent will purchase only enough components to meet the need from the expected production in Assumption 2. The prices of these components will be the same as recently observed prices.
4. Computers held in inventory at the end of 12 days are of no value.
5. The agent is able to accurately predict the probability of winning an order given the bid price.

Assumption 1 means that our agent has a fixed set of resources to work with when selecting bids. This is a reasonable assumption due to the fact that our agent (and many other agents from the competition) tries to carry a large component inventory and tends to place long-term rather than short-term component orders. Because the prices paid for components in inventory are sunk costs, our agent will only consider replacement costs when determining the cost of producing a computer. These costs can be determined from Assumptions 2 and 3 by projecting future component use, deciding whether the components used over the next 12 days will need to be replaced, and determining how much this will cost per component. If the current inventory of a component exceeds the projected use, then that component's replacement cost will be zero. Assumption 4 means that our agent should be willing to use all computers in inventory and all upcoming production cycles for computers that will be delivered over the next 12 days. Assumption 5 simply tells us that our agent has access to the bid acceptance functions we tried to learn in the previous section.

As a result of these assumptions, our agent is essentially pretending that it is acting in a static environment, and this could lead to suboptimal behavior if future game circumstances change. For example, if computer prices are currently low due to low demand, but the trend of customer demand is increasing, then it might be wise to hold on to computers in inventory for later sale, violating assumption 4. The ability to more accurately predict the future values of components and computers would be valuable, but we leave this to future work.

The profit our agent obtains over the next 12 days depends not only on the RFQs being bid on on the current day, but also on RFQs that will be received on later days for computers due during the period. If we were to ignore these future RFQs when selecting the current day's bids, our agent might plan to use up all available production resources on the current RFQs, leaving it unable to bid on future RFQs. One way to address this issue would be to restrict the resources available to the agent for production of the computers being bid on currently. This is the method used by [6]. We instead take the approach of predicting the RFQs that our agent will receive for computers due during the period, and coming up with bids for these RFQs at the same time as the actual RFQs from the current day. Future RFQs are randomly generated according to the parameters given in the game specification and our current estimate of the level of customer demand and its trend. This has the effect of causing our agent to decide which resources to reserve for future RFQs, and limited testing suggests that our agent performs better when using this method than when we explicitly restrict the resources available.

### 5.2 Optimization Method

We now have an optimization problem with the following inputs:

- The agent's current computer orders
- The resources available to the agent over the next 12 days: production cycles, computers and components currently in inventory, and expected future deliveries of components
- A cost associated with each computer representing the expected replacement costs of its components
- A set of RFQs for computers due over the next 12 days, including both the current day's actual RFQs and predicted future RFQs.

Our goal is to find the set of bids that maximizes expected profit on these RFQs and existing orders. Those bids representing actual RFQs for the current day will then be offered to customers.

We make the assumption that we will always want our agent to fill existing orders if possible, and so the agent begins by scheduling the production necessary to fill these orders. This leaves our agent with a reduced set of resources and means that it only needs to concern itself with the expected profit from RFQs.

If we were considering only a single auction and had no resource constraints, the expected profit resulting from a particular bid price would be:

$$Expected\ profit = P(order|price) * (price - cost) \tag{1}$$

The optimal bid would be the value that maximized this quantity.

Computing the expected profit from a set of bids when resource constraints are considered is much more difficult, however, because the profit from each auction cannot be computed independently. For each possible outcome of the auctions in which it is not possible to fill all orders, the profit obtained depends on the agent's production and delivery strategy. For any nontrivial production and delivery strategy, precise calculation of the expected profit would likely require separate consideration of a number of possible auction outcomes that is exponential in the number of auctions. If we were guaranteed that we would be able to fill all orders, we would not have this problem. The expected profit from each auction could be computed independently, and we would have:

$$Expected\ profit = \sum_{i\ \epsilon\ all\ auctions} P(order_i|price_i) * (price_i - cost_i) \tag{2}$$

Our bidding heuristic is based on the assumption that the expected number of computers ordered for each RFQ will be the actual number ordered. In other words, we pretend that it is possible to win a part of an order, so that instead of winning an entire order with probability $p$, we win a fraction $p$ of an order with probability 1. This assumption greatly simplifies the consideration of filling orders, since we now have only one auction outcome to consider, while leaving the formulation of expected profit unchanged. As long as it is possible to fill the partial orders, (2) will hold, where the probability term now refers to the fraction of the order won. It would appear that this approach could lead to unfilled orders when the agent wins more orders than expected, but in practice, this is not generally a problem. Most of the RFQs being bid on are the predicted RFQs that will be received on future days, and so the agent can modify its future bidding behavior to correct for an unexpectedly high number of orders resulting from the current day's RFQs. The agent can also set aside a small number of completed computers in inventory to serve as a buffer to prevent penalties in case any problems remain. When using this bidding strategy, our agent indeed tends to have very few late or missed deliveries.

By using this notion of partial orders, we can transform the problem of bid selection into the problem of finding the most profitable set of partial orders that can be filled with the resources available. Although this problem lends itself to standard optimization methods, our choice of method is constrained by the limit of 15 seconds per simulated game day and by the size of the problem (there may be over a thousand RFQs to consider). We use a greedy production scheduler that tries to utilize production resources as profitably as possible. All bids are initially set to be just above the reserve price, which means we begin with no orders. The production scheduler then chooses an RFQ and an amount to lower its bid by, resulting in an increased partial order for that RFQ. The scheduler simulates filling this increase by scheduling its production as late as possible, taking completed computers from inventory if production is not possible. This process is repeated until no more production is possible or no bid can be reduced without reducing the expected profit.

Because we are working with resource constraints, the goal of the greedy production scheduler at each step is to obtain as large an increase in profit as possible while expending as few production resources as possible. To illustrate how this is done, consider Figures 4, 5, and 6. Fig. 4 represents the predicted probability of winning an auction as a function of the bid price, or alternatively, the fraction of the auction we assume we will win at each bid price. Fig. 5 shows the expected profit at each price, found using (1). Now suppose that our current bid is 1500, and we are considering lowering this bid. We would like to find the bid decrease that produces the largest increase in profit per additional computer ordered. This quantity is equal to $(Profit(x) - Profit(1500))/(Probability(x) - Probability(1500))$ for $x < 1500$ and is graphed in Fig. 6. From the graph, we can see that the optimal decision is to lower the bid to about 850. At each step, the production scheduler performs this analysis to find the bid reduction that will produce the largest increase in profit per additional computer for each RFQ, and chooses the RFQ for which this value is the largest.

In many cases, the most limited resource is production cycles. In such cases, the increase in profit per cycle used is a better measure of the desirability of a partial order than the increase in profit per additional computer, so we divide the latter quantity by the number of cycles required to produce the type of computer requested by the RFQ and use the resulting values to choose which RFQ should be considered next. We consider cycles to be the limiting factor whenever the previous day's production used more than 90% of the available cycles to produce computers used to fill orders (as opposed to computers produced using spare cycles in order to build up inventory).

The range of possible bid prices is discretized for the sake of efficiency. Even with fairly fine granularity, this bidding heuristic produces a set of bids in significantly less time than the 15 seconds allowed per simulated game day. Attempts to use local search methods to improve the bids found yielded almost no increase in profit, suggesting at the very least that our greedy method tends to find local minima. The complete bidding heuristic is summarized in Table 1.

# 6 Agent Performance

In this section, we evaluate the effectiveness of our learning approach and bidding method when used as part of a complete agent in TAC SCM gameplay. We do this through a series of experiments in which agents using different combinations of bidding methods and prediction methods play against each other repeatedly.

## 6.1 Agent Design

For each agent, production and delivery are handled by a greedy production scheduler that gives near-optimal performance in practice. In order to isolate

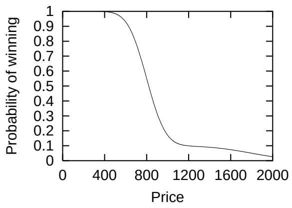


**Fig. 4.**

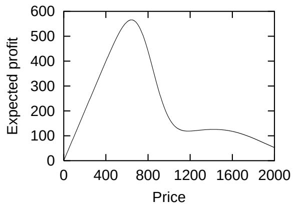


**Fig. 5.**

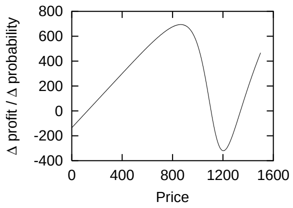


**Fig. 6.**

**Table 1.** The bidding heuristic

- For each RFQ, compute both the probability of winning and the expected profit as a function of price
- Set the bid for each RFQ to be just above the reserve price
- Repeat until no RFQs are left in the list of RFQs to be considered:
  - For each RFQ, find the bid lower than the current bid that produces the largest increase in profit per additional computer ordered (or per additional cycle required during periods of high factory utilization)
  - Choose the RFQ and bid that produce the largest increase.
  - Try to schedule production of the partial order resulting from lowering the bid. If it cannot be scheduled, remove the RFQ from the list.
  - If the production was scheduled, but no further decrease in the bid will lead to an increase in profit, remove the RFQ from the list.
- Return the final bid for each RFQ.

the effects of bidding, we modified the game settings to allow each agent to receive an effectively unlimited quantity of each component on the 15th game day at no cost, eliminating the need for a strategy for purchasing components from suppliers. This is not entirely unrealistic, as many agents in the competition actually ordered the majority of their components on the first game day [12]. Agents were only allowed to carry up to 200 of each type of computer in inventory, to prevent them from using their limitless components to build up large computer inventories during periods of low customer demand. The effect of this limitation was to increase the responsiveness of computer prices to changes in demand, creating a more dynamic and interesting game scenario.

The agents differ in the bidding methods used and the predictions of bid acceptance probability. The bidding methods used are

- Bidder: The bidding method presented in Sect. 5.
- OldBidder: A previously developed hill climbing bidding method [2].

and the bid acceptance prediction methods are

- Learning: Learning as described in Sect. 4.
- Heuristic: The heuristic described in Sect. 4.2.
- OldHeuristic: A previously developed prediction heuristic [2]. Testing on game data has shown this heuristic to be less accurate than Heuristic.

OldBidder and OldHeuristic are taken from TacTex, our entry in the 2003 TAC SCM competition, and are described in detail in [2]. Each of the six combinations of bidding methods and prediction methods is used by one agent.

### 6.2 Experimental Setup

Three rounds of 30 games were played between the six agents. During the first round, the agents labeled as using Learning actually used the same heuristic as Heuristic. The game logs from the first round were then used to train sets of predictors to be used by the Learning agents in the second round. For both agents, we trained a separate predictor for each of the 26 price points between 0 and 1.25 times the base price spaced at an interval of 0.05, using BoosTexter as the learning algorithm and using 10% of the available data (about 100,000 instances). Because each learning agent is trying to outbid only the other agents and not itself, its own bids were ignored when determining the winning bid for each training instance. The functions mapping bids to probabilities of acceptance are created from the 26 predictions by enforcing a monotonicity constraint as described in [9], with the added step of setting all probabilities for bids above the reserve price to 0. A second round of games was then played.

In the third round, the Learning agents used a set of predictors that had been trained on the logs from the semifinal and final rounds of the 2003 TAC SCM competition. The purpose of this was to determine how well the predictors would generalize to a different set of agents.

### 6.3 Results

The results are presented in Table 2. The average relative score of each agent is given along with the standard deviation. An agent's relative score in a game is its score minus the average score of all agents for that game. The average score over all agents and games in each round was around $80 million. Because all agents were initially given sufficient components to last the whole game, no component costs are included in any of the scores presented.

From the results of the first round, we can see that agents using Bidder outperform the agents using OldBidder, and that for each bidding method, agents using Heuristic outperform agents using OldHeuristic. This is the expected result.

The results of the second round show exactly what we had hoped to see: using learning significantly improved agent performance. Bidder/Learner outscored Bidder/Heuristic in all but one game and by an average margin of $6 million.

In the third round, the agents using Learning still showed a performance improvement, but by a smaller margin. Considering that the predictors used by Learning were trained on games involving a completely different set of agents, and a somewhat different game scenario (i.e., a limited component supply), this result is very promising. In actual competition, we might not have access to games involving only the agents we are competing against, and this experiment suggests that learning could still be successfully applied in such a case.

## 7 Future Work and Conclusion

In this paper, we considered the problem faced by an agent acting in a supply chain that must bid in simultaneous reverse auctions to win orders from customers. Using TAC SCM as a test domain, we presented a learning approach to the task of predicting the probability of bid acceptance, and we presented a heuristic bidding method that uses these predictions. A comparison of learning algorithms showed that M5 regression trees and BoosTexter result in similar prediction accuracy when testing and training data come from separate games. When used as part of a complete agent, learned predictors were shown to provide a significant improvement in performance over heuristic predictors.

One important result demonstrated was that the learned predictors generalize well to new situations, both in terms of prediction accuracy and of agent

**Table 2.** Average relative score (in millions)

| *Agent* | *Relative Score* | | |
|---|---|---|---|
| | *Round 1* | *Round 2* | *Round 3* |
| Bidder/Learning | 6.10 ± .28 | 9.04 ± .3 | 6.49 ± .73 |
| Bidder/Heuristic | 6.13 ± .28 | 2.95 ± .42 | 5.20 ± .57 |
| Bidder/OldHeuristic | 2.2 ± .30 | -.31 ± .42 | 1.37 ± .40 |
| OldBidder/Learning | -4.17 ± .22 | 1.60 ± .55 | -1.80 ± .70 |
| OldBidder/Heuristic | -4.21 ± .24 | -5.87 ± .30 | -4.54 ± .53 |
| OldBidder/OldHeuristic | -6.09 ± .39 | -7.41 ± .48 | -6.72 ± .46 |

performance. This gives us hope that our learning approach can be used successfully in competition when facing different sets of agents or agents that change their behavior over time.

There are several possible ways in which predictions could be improved. The results of Sect. 4.3 suggest that acquiring data from a variety of situations might aid in training a more robust predictor. Further experiments could determine the best combinations of data for an agent to use. Also, additional information available to an agent could be included as features, such as knowledge of the availability and prices of components. Finally, an agent could make use of its knowledge of auction results during a game to make on-line improvements to its predictors. Boosting-based predictors would lend themselves well to this approach, since making incremental modifications to the existing predictors would be straightforward.

The heuristic bidding method presented appears to work well, but needs to be made less myopic. This could be done by developing better estimates of the future values of components and computers in inventory, in order to allow more informed decisions of whether to hold on to them for later use.

## Acknowledgments

This research was supported in part by NSF CAREER award IIS-0237699.

## References

1. Sadeh, N., Arunachalam, R., Eriksson, J., Finne, N., Janson, S.: TAC-03 a supply-chain trading competition. AI Magazine (2003)
2. Pardoe, D., Stone, P.: TacTex-03: A supply chain management agent. SIGecom Exchanges **4** (2004) 19–28
3. Arunachalam, R., Eriksson, J., Finne, N., Janson, S., Sadeh, N.: The TAC supply chain management game. Technical report, Swedish Institute of Computer Science (2003) Draft version 0.62.
4. Papaioannou, V., Cassaigne, N.: A critical analysis of bid pricing models and support tool. In: IEEE International Conference on Systems, Man and Cybernetics, Piscataway, NJ (2000)
5. Lawrence, R.D.: A machine-learning approach to optimal bid pricing. In: Proceedings of the Eighth INFORMS Computing Society Conference on Optimization and Computation in the Network Era, Arizona (2003)
6. Benisch, M., Greenwald, A., Grypari, I., Lederman, R., Naroditskiy, V., Tschantz, M.: Botticelli: A supply chain management agent designed to optimize under uncertainty. SIGecom Exchanges **4** (2004) 29–37
7. Kiekintveld, C., Wellman, M., Singh, S., Estelle, J., Vorobeychik, Y., Soni, V., Rudary, M.: Distributed feedback control for decision making on supply chains. In: International Conference on Automated Planning and Scheduling. (2004)
8. Dahlgren, E., Wurman, P.: PackaTAC: A conservative trading agent. SIGecom Exchanges **4** (2004) 38–45
9. Schapire, R.E., Stone, P., McAllester, D., Littman, M.L., Csirik, J.A.: Modeling auction price uncertainty using boosting-based conditional density estimation. In: Proceedings of the Nineteenth International Conference on Machine Learning. (2002)
10. Schapire, R.E., Singer, Y.: BoosTexter: A boosting-based system for text categorization. Machine Learning **39** (2000) 135–168
11. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann (1999)
12. Estelle, J., Vorobeychik, Y., Wellman, M., Singh, S., Kiekintveld, C., Soni, V.: Strategic interactions in a supply chain game. Technical report, University of Michigan (2003)