A Simple Agent for Supply Chain Management

Brian Farrell and Danny Loffredo

December 7th, 2006

1 Introduction

The Trading Agent Competition Supply Chain Management (TAC SCM) game is a competitive testbed for software agents designed to perform supply chain management. While not a perfect simulation of reality, the game is sufficiently complex to encourage the development of complex supply chain management strategies.

A TAC SCM game consists of 220 simulated days. On each day, customers send requests for quotes (RFQs) for new computers to the agents. The agents respond to these RFQs with offers that can be accepted or rejected by the customers. The agents build computers using parts bought from suppliers. The agents send RFQs to the suppliers, specifying a desired price and due date. The suppliers can then make an offer on the RFQs, which the agents can accept or reject. More detailed information about the game can be found in the official specification document [1]. Stated simply, there are two main objectives in the game: to minimize component procurement costs, and to maximize computer sales prices.

In this paper, we describe Simplicity, an agent designed to compete in TAC SCM. The agent was designed using a simple approach in order to see how it would fare against more complex implementations from previous TAC SCM tournaments. The paper is organized as follows. First, we explain the design and implementation of the agent. Next, we analyze the success of our strategies. Finally, we suggest improvements that could be made in future work.

2 Implementation

Our chief goal in designing Simplicity was to create a simple agent. While the limited timeframe of the project necessitated a simple design, there are also other benefits to designing a simple agent. However, there are many benefits of a simple agent design. All other things being equal, a simple agent would be preferred over one that is more complex. It is easier to understand, implement, and debug. We hoped to show whether a simple agent could still do reasonably well when compared to more complex approaches. Even if the agent did not perform as well as some of the other agents entered in the TAC SCM

tournament, it could still be possible to describe it as successful if it performed nearly as well as the other agents. There are very likely to be tradeoffs between agent performance and complexity. The goal of Simplicity was to see if much of the complexity could be reduced while still maintaining a reasonable amount of performance.

Another advantage Simplicity has over other agents is that it does not use any data from previous games. Several of the top performing agents in the 2005 TAC SCM tournament used data from previous games in some fashion [3][4]. While this data will always be easily obtained in this competition, it may be harder or impossible to obtain in other domains, so there is some benefit to an agent that does not need past data. However, an agent that does not require past data is not necessarily more robust to drastic changes in the market. Even though the agent does not use data from the past, it was still tested in market conditions similar to the ones it would face in the competition. It is hard to say whether it would perform well if some facets of the competition were drastically changed.

An important idea that informed the design was the possibility of selling computers at a loss. There is a bias towards low-demand games [5]. During periods of low demand, prices can drop below the level at which sales are profitable. An intelligent agent must know when it is better off accumulating inventory and when it is better off selling computers. In order to make this decision reliably, the agent needs an accurate measure of its costs.

The tasks of the agent are split into two modules: the Demand Manager and the Supply Manager. The Demand Manager handles customer RFQs and factory production. It decides which RFQs to bid on, and what prices to pay. It then schedules factory production and deliveries, and projects how many components it will need in the future to maintain production. The Supply Manager deals with suppliers. It uses the usage projections from the Demand Manager to buy inventory.

2.1 Supply Manager

The Supply Manager's main goal when placing normal orders is to be very flexible. The price offered by a supplier is a function of its committed capacity, so prices are very much affected by the actions of other agents. An agent that always buys components with a lead time of 5 days will do well when playing against agents that all use long-term buying strategies, but will perform poorly when playing against agents with short-term buying strategies. Therefore, it is important to be as flexible as possible when considering component lead times.

The Supply Manager sends five RFQs to suppliers each day. First, it sends an RFQ to procure any components necessary to prevent shortages in the next twenty days. Next, it sends an RFQ for components with either an extremely short or long lead time. Finally, it uses whatever remaining RFQs it has to send zero-quantity RFQs to suppliers which it uses to model the suppliers' production to estimate component costs in the future.

2.1.1 Normal RFQs

Each day, the Supply Manager uses the list of projected component use supplied by the Demand Manager to determine when its inventory will drop below a specified level. This level was set at 800 for non-CPU components and 400 for CPU components. If the projected inventory level after twenty days is still above the threshold, no normal orders are placed. If not, the Supply Manager looks at all of the days that it has price predictions for that are before the day that inventory becomes too low. It plans to buy slightly more than it would need to get inventory levels back to the threshold. This has the effect of keeping component lead times closer to the twenty-day cutoff. Once the Supply Manager finds the day with the lowest predicted price, it sends an RFQ with a reserve price slightly greater than the predicted price. Since large procurements by other agents could drastically change the supplier's offer price, and the agent uses RFQs that are up to three days old in price prediction, a reserve price is necessary. It keeps the agent from paying a price much greater than it expected. Because of this reserve price, the agent can simply accept every offer that it gets as a result of the RFQ.

2.1.2 Short- and Long-Term RFQs

While the strategy for procuring components via normal orders is fairly flexible, it ignores dates less than four days into the future and greater than twenty days into the future. In order to make the agent's procurement strategy as flexible as possible, these dates must be considered as well. Thus, the agent should be able to procure components at a low cost no matter what strategy its competitors are using. Each day, the agent sends either a short term order with a lead time of two days (the minimum), or a long term order with a lead time of 37 days.[1] The order was always for 25 CPU components or 50 non-CPU components. Rather than trying to predict the selling price as in the normal orders, the Supply Manager simply sets the reserve price to be slightly below the agent's average procurement cost for that component. This way, if the supplier makes an offer, the agent can accept it unconditionally because it will bring average costs down. Since the efficacy of a procurement strategy depends on the other agents playing in the game, it is possible that the short-term RFQs may get offers more often than the long-term RFQs, or vice versa. In order to take advantage of situations where this occurs, the Supply Manager chooses which type of RFQ to send based on its success at getting offers thus far in the game. If the other agents are all doing long-term purchasing, the Supply Manager will send mostly short-term RFQs because they will be more successful.

[1] The lead time of 37 days was chosen arbitrarily; the motivation for choosing an unusual number was to lessen the possibility that other agents would try to procure with the same lead time.

2.1.3 Zero-Quantity RFQs

Since the Supply Manager looks twenty days into the future for each component purchase, probe RFQs are sent 20 days into the future, and every four days before that until the Supply Manager has used all five of its RFQs. The Supply Manager uses these probes to build a model of the supplier. Since the RFQs sent on each day are spaced four days apart, the supplier model can look at a window of 12-16 days (depending on whether it was able to send three or four probes over the last several days) when it does price prediction. Using RFQs that are a few days old allows the Supply Manager to choose from a larger number of days when procuring components. If it only used RFQs from the previous day, it would only be able to choose from five days.

2.1.4 Price-Increasing Orders

Since the Supply Manager looks twenty days into the future when planning procurement, it should not need to buy any parts after day 200. Rather than just sit idle, the Supply Manager sends RFQs to suppliers for parts that it does not intend to buy. This will drive up prices for the agents that still need to procure components. Either the agents will be forced to buy components for a very high price, or they will not purchase any components because the price is too high. It is important to choose the right order size when sending orders to drive up prices. Since the prices offered by suppliers are an increasing function of their committed capacity, a larger order will drive up prices more. However, suppliers consider agents' RFQs in order of reputation. If the agent's reputation dips below 1.0, its large orders will not affect prices for the other agents. The agent determines how many orders it can make without lowering its reputation below 1.0, and plans to use up all of these orders before the end of the game.

2.2 Demand Manager

The Demand Manager has three main tasks: produce or deliver computers for existing orders, make offers on new RFQs, and send the next day's production schedule to the factory. To reduce late delivery penalties, our delivery scheduler first tries to fulfill late orders, then current orders, and eventually early orders. Each order type is sorted by profit, so that the highest profit orders are fulfilled first. New RFQs are also sorted by estimated profit. An offer is submitted for any profitable RFQ in which the completed computers are already in inventory, or can be produced in time to deliver by the desired due date. Currently the agent assumes that all offers sent out will be accepted. Finally, the factory scheduler uses a greedy algorithm to produce computers required for the next day's deliveries. If there is left over factory capacity, then future production is moved ahead of schedule.

2.2.1 Determining Offer Price

One of the key decisions that the Demand Manager makes is setting the offer price for completed computers. We decided to perform linear regression on the last n sale prices, where `n = 10` was selected as a good balance between responsiveness and stability. The history of lowest selling prices and highest selling prices were both valuable, depending on the quality of our competitors. Against good agents, the highest sale price will be reasonable, and provides a good price target to slip under. Against poor agents or in extreme market conditions, the lowest sale price provides a lower bound to come in above. In cases where we need to sell computers as quickly as possible, like when inventory levels are too high at the end of the game, we use regression on the history of low prices to set our base sale price. Otherwise, we use regression on the history of highest selling prices.

Two simple modifications were made to this basic regression strategy. First, if there is no difference between past high sell price and low price, this indicates that there may be room to slightly increase our sale price. Second, furthering our philosophy of only selling for profit, we set minimum offer prices. The minimum offer price was usually equal to our cost, as calculated by the moving average. However, if costs were unusually high (above `0.7*basePrice`), we set the minimum offer price to be `0.7*basePrice`. This would result in selling computers at a loss, but costs at that level were rarely observed. It was put in place to keep the agent from possibly stopping all production due to several days of very high costs. The only time we do not set minimum prices is during a ramp down phase for the last 20 days of game time. During this time, the agent is essentially dumping as much inventory as possible before the game ends.

In highly competitive games, it is possible that our agent will sell a relatively small number of computers over the course of several days. Two mechanisms were implemented to prevent overstocking inventory in these cases. First, once a threshold of one computer type is reached, the agent won't order replacement components for new purchase orders. Replacement components are only ordered when inventory levels drop below the threshold of 250 computers. Secondly, the Demand Manager estimates future component usage as described in the next section. This keeps component inventory levels reasonable without having to explicitly calculate the chance each computer offer will be accepted.

2.2.2 Projecting Future Component Use

The only way in which the Demand Manager communicates with the Supply Manager is through its projections of future component use. The Supply Manager uses these productions to determine when it needs to buy more components. For each component, the usage projector regresses usage as a function of demand over the last ten days. More specifically, usage on a day `d` is defined as the amount of a component used in production on day `d`. Demand is defined as the number of RFQs sent from customers on day `d` for computers containing that component.

Once the projector has performed this regression, it extrapolates for each day, twenty days into the future. Since the number of RFQs on some day in the future is not known, the agent uses a prediction from the DeepMaize demand predictor. The demand predictor attempts to model the customer demand function as given in the TAC SCM specification. It is described in more detail in [2].

3 Results

3.1 Empirical evaluations

3.1.1 Supply Manager

The Supply Manager was able to get competitive prices for its component orders. In trial games against agents from the TAC SCM 2005 tournament, the agent's component costs were similar to those of other agents. It is not always instructive to simply rank procurement strategies by lowest cost; an agent with a long-term procurement strategy will get lower costs, but will lose some flexibility in responding to demand. We feel that Simplicity's strategy allowed for low costs while still being very responsive to changes in demand.

We ran forty-eight games with Simplicity and a copy of Simplicity that did not make short- and long-term orders. The short- and long-term ordering strategy resulted in lower costs for CPU components. This result was significant with 95% confidence according to a paired t-test. There was a negligible difference in costs for the other components. While we are not sure why the strategy was effective for CPUs and not for other components, there are definite differences between CPUs and other components because each CPU can only be purchased from one supplier, while other components can be purchased from two suppliers. More investigation would be required to determine why the strategy was not effective for non-CPU components.

The endgame strategy of sending large RFQs that the agent would not accept did not affect any of the TAC SCM agents we tested against, because they generally had enough components to last until the end of the game. However, we know that the strategy can be effective, given the right agent: when Simplicity competes in a game with "dummy" agents that only buy components when they receive an order, the "dummy" agents were observed spending as much as 40 times the base price for their components. Since the agent wouldn't be sending any RFQs during the last days of the game, it makes sense to leave this strategy in because in the worst case, it has no effect on the game, and in the best case, it causes competitors' costs to skyrocket.

3.1.2 Demand Manager

Table 1 shows an experimental comparison of TACStarterAgent and our agent. TACStarterAgent is a skeleton agent that has all the required functionality to play in a game of TAC SCM, but utilizes bare strategies. The techniques

Agent	Average Selling Price	Number Sold
TACStarterAgent	1696.58	3222
Simplicity	1926.77	4816

Table 1: Comparison of Average Selling Price

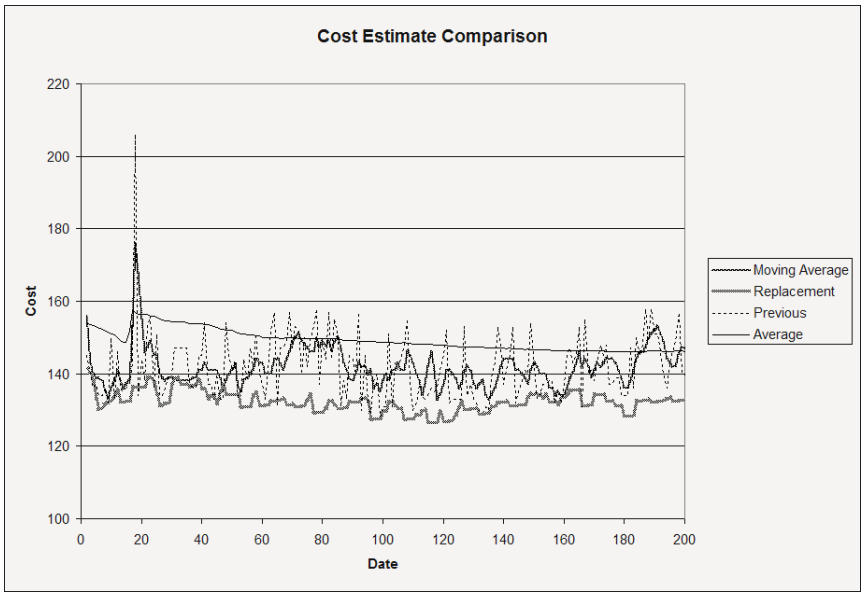

Figure 1: Comparison of different techniques for estimating component costs

implemented in our Demand Manager give Simplicity a 13% higher average selling price over the starter agent, in addition to a 49% increase in total number of computers sold.

Since we strive to only sell computers at a profit, it is important to accurately calculate the cost to build each computer. We considered four techniques to estimate each component's cost:

1. previous price - the price the component was last purchased at
2. average price - average price paid for the component
3. moving average - average price over the last `n` days
4. replacement cost - estimated cost to replace the component in the next 20 days

Figure 1 shows how the four techniques differ in practice. Average price remains steady and does not follow short-term trends, while previous price is the most volatile of the four. Replacement cost tends to underestimate the

Agent	Average Cost
Simplicity	655
StormFront	666
JAgent	679
redbull	697
garfield	744

Table 2: Comparison of Average Costs for Pintel 2.0 GHz during the tournament

component cost, since it is based on a model of future demand rather than the current prices. Ultimately we decided to use a moving average as our metric, because it accurately reflected current market conditions yet provided more stability than the previous price.

3.2 Class Tournament

Simplicity finished in second place among five agents in the class competition. We looked at component costs for Pintel 2.0 GHz CPUs to determine how successful the Supply Manager was. Table 2 shows the average cost over the sixteen games of the tournament for Pintel 2.0 GHz CPUs. Simplicity had the lowest average cost over the entire tournament, and was the agent with the lowest cost in nine of sixteen games.

One of the successes of our agent in the tournament was the ramping down strategy in the last 20 days of game time. During this time, the agent attempts to drive up component prices by placing large component orders at high prices, and will begin dumping its own inventory of completed computers. Due to competitive pricing on the part of several of the tournament participants, Simplicity often had high inventory levels when the ramp down began. One measure of the success of this strategy is market share before and during the ramp down. For days 0 through 199, Simplicity achieved an average market share of 20% of the total demand met. During the endgame, for days 200-219 the agent increased its average market share to 29% of the total demand met. In some of the tournament games our agent's score didn't even go positive until all its inventory was dumped at the end of the game.

4 Conclusions

4.1 Limitations and Future Work

Even though we achieved our goal of creating a simple agent that performs close to more complex agents, there are several areas that could use improvement. In the Supply Manager, component orders can be split between the two suppliers rather than getting the entire order from one. This change would lead to overall lower costs, but would require figuring out the optimal amount to request from

each supplier. In addition, short and long term orders are set for a fixed amount; instead, this could be a function of the acceptance rate. In the Demand Manager, the projected component use relies on the indirect relationship between the number of components used in a day's production and the estimated demand. This technique could be more robust and quicker to react to changing market conditions if some other, more direct, relationship were found to estimate component usage. Also, the Demand Manager could utilize the demand predictor to attempt to find peaks in demand, and try to unload inventory at those times. One important change we did not have time to implement was to calculate the probability that each order would be accepted. This change would effect several components of the agent, including the offer price, estimated component usage, and factory scheduler. Finally, the agent has several hand-tuned values, like the inventory caps, that could be adaptively adjusted or learned over the course of several games.

4.2 Summary

Despite its shortcomings, our agent does manage to perform reasonably well against more complicated agents. We achieved second place in the class competition by maintaining high sale prices, buying components cheaply, and dumping inventory in the endgame. Our core strategies were to not sell computers at a loss, and to stay as flexible as possible with our procurement. By creating an agent which is simple yet powerful, we have a robust, easy to debug agent, as well as an good starting point for more sophisticated strategies.

References

[1] J. Collins, R. Arunachalam, N. Sadeh, J. Eriksson, N. Finne, and S. Janson. The supply chain management game for the 2006 trading agent competition.

[2] C. Kiekintveld, M. P. Wellman, S. Singh, J. Estelle, Y. Vorobeychik, V. Soni, and M. Rudary. Distributed feedback control for decision making on supply chains.

[3] D. Pardoe and P. Stone. TacTex-2005: A champion supply chain management agent. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 1489–94, July 2006.

[4] P. Toulis, D. Kehagias, and P. A. Mitkas. Mertacor: a successful autonomous trading agent. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1191–1198, New York, NY, USA, 2006. ACM Press.

[5] M. P. Wellman, J. Estelle, S. Singh, Y. Vorobeychik, C. Kiekintveld, and V. Soni. Strategic interactions in a supply chain game. *Computational Intelligence*, 21(1):1–26, February 2005.