PackaTAC: A Conservative Trading Agent

Erik Dahlgren
dahlgren@home.se
and
Peter R. Wurman
wurman@ncsu.edu
Department of Computer Science
North Carolina State University

The Supply Chain Management game presents a challenging manufacturing scenario where agents compete for customer demand and supplies needed to produce the demanded products. The entry of North Carolina State University, *PackaTAC*, is a relatively simple agent that plays a conservative, low-risk strategy. Its aim is to never bid on more orders than it can handle, while maintaining low inventory levels and aspiring to full factory utilization. The agent relies on the fact that it will have ten opportunities to win contracts for a given production day, and constantly adjusts its profit margins to track the market prices. Its low risk approach was chosen to combat the mutually destructive strategies that were emerging during the qualifying rounds. Despite PackaTAC's simple approach, it performed quite well in the 2003 competition held at the International Joint Conference on Artificial Intelligence in Acapulco.

Categories and Subject Descriptors: K.4.4 [**Computers and Society**]: Electronic Commerce

Additional Key Words and Phrases: Trading Agents

1. INTRODUCTION

TAC-SCM is a contest in which six agents, each representing a computer factory, compete in a common market place for suppliers and customer orders. The customers issue orders every day, which among other things specify the PC type and number of computers they want. There is also a due date associated with each order. If an order is not delivered by the agreed upon date the agent is penalized. In order to produce computers, the agents need to buy the components from their respective supplier(s).

Each game in the competition lasts for 220 simulated days, long enough to create three distinct game phases: startup, middle, and end game. Each day, the agents have to decide which components to order from the suppliers and which customer orders to bid on, as well as plan their production and shipping schedules. In the end, the agent with the highest profit is the winner. See [Arunachalam et al. 2003] for the game specifications.[1]

[1]A lot of information about TAC-SCM, and also TAC-Classic, can be found at the official website at

The agent from North Carolina State University, *PackaTAC*, used a conservative heuristic strategy. Part of this strategy is to never bid on more orders than it can handle, in order to minimize penalties, while still maintaining full factory utilization. This is covered in greater detail in Sections 3 and 4. It also tries to keep low supply inventory levels to allow for changes in demand and supply of computers and parts, as described in Section 2. In the final round of the IJCAI competition, after the field was winnowed to six agents, PackaTAC finished fifth.

2. OBTAINING SUPPLIES

One of the more difficult problems in the game is to make sure that supplies are available at the time they are needed for production. This is a hard problem because, although the suppliers make promises based on a nominal production level of 500 units per day, their actual production level is determined by a random walk. Thus, when the supplier produces less than it planned, it delays the deliveries to the agents. The server generates a market report every 20 days that provides information about the different components, such as the amounts produced and the individual prices. However it does not specify whether each supplier is under- or over-producing. It may be possible for an agent to maintain a model of the supplier in order to better gain this information.

One important factor is the price of supplies, which ranges between $[0.5 * \mathrm{basePrice}; 1 * \mathrm{basePrice}]$. The price depends on demand, where greater demand on the part of the agents leads to high prices. On the first day, because there is no previous demand, the price is $0.5 * \mathrm{basePrice}$. Because this is the lowest the prices can ever be, many agents were built on strategies that included ordering all the supplies needed for the whole game on the first day. We call these strategies *big-order* strategies.

2.1 Big-order Strategy

One strategy was to buy most or all of the supplies for the whole game on the first day. If you can anticipate the quantity of supplies that you will need during the course of the game, buying them on the first day is the cheapest. In some games, the price stayed considerably higher than the first day low, and by using this strategy an agent would not have to buy much at the higher prices. This is similar to the strategy in the TAC Classic game where agents bought flight tickets early to get low prices [Cheng et al. 2002; Wellman et al. 2002]. A side effect of this strategy is that the big orders can block the suppliers from delivering to any agents while the big orders are being filled.

In games where multiple agents play the big-order strategy they may have to wait for the other agents' large orders, and as a result their own supplies get delivered very late in the game. Frequently in these games, agents playing this strategy incurred large negative scores because they either ended up with large quantities of unused inventory or competed away profits. Thus, we concluded that the big-order strategy was mutually destructive.

It was quite evident early on in the competition that some agents used this kind of strategy, including the winning agent, *RedAgent* [Duguay et al. 2003]. A few agents would only order components on the first day, and would never again communicate with the suppliers. Later on during the competition, one agent, *Deep Maize*, adopted a strategy in which they tried to block the other agents' first day orders by requesting and declining large amounts of supplies. This strategy seemed to work fairly well [Estelle et al. 2003].

`http://www.sics.se/tac`

2.2 PackaTAC's Strategy

We observed agents that played the big-order strategy, and noticed that in low demand games they tended to perform very poorly and ended up with very bad scores. Because of this, we decided to not use the big-order strategy for *PackaTAC*. Instead, on the first day PackaTAC would order an *initial quantity* for each component in six deliveries spread throughout the first half of the game. In the finals, the initial quantity was set to 1,800. In contrast, some of the agents that played the high risk strategy ordered more than 30,000 units of each component on the first day. One reason this strategy worked well, and the big-order strategy did not, is because there is a clear bias for low demand games [Estelle et al. 2003].

After this initial ordering, the agent tried to maintain a *target* inventory level. This was set to 1500, which would last for at least nine days of production. Whenever the inventory for any supply went below the target level PackaTAC would reorder that supply. However, to prevent PackaTAC from over-ordering in the case where it was low on supplies, but the suppliers were busy with other orders, there was a *back order limit*. This is the maximum amount that is allowed to be in orders not yet delivered. This prevented the agent from ordering too much when delivery of supplies was delayed.

Also, as part of our conservative approach, PackaTAC did not count on promised deliveries when bidding on orders. If an order could not be produced with current inventory levels, the agent would not bid on it, even if the missing supplies were supposed to arrive the next day.

All the requested delivery dates for the initial large orders were in the first half of the game, however it was evident during the final rounds that this date frequently got pushed back by the supplier until the last few days of the games. When this occurred, the agent performed very poorly since it was often out of one type of component, and thus did not have the capability to build any computers. Due to a design flaw, it also frequently failed to issue smaller orders in the mean time. The back order limit was designed to take into account these large initial orders so they would not fill up the limit. However, the logic did not properly handle delivery dates that strayed too far from the original plan. If most of the initial orders had a due date that differed from the plan they would fill up the back order limit, and the agent would not order more supplies even if it was out. As the game carried on, the requests for supplies went down, and other agents performed extremely well while PackaTAC was stuck waiting. There was a quick attempt to alleviate this problem right before the final round, but it did not seem to have any significant impact on the performance of the agent.

3. SCHEDULING

One of the main challenges an agent faces in the TAC game is scheduling. The scheduling problem is how to allocate supply resources and factory time—what to produce, and when to produce it. For PackaTAC we first tried a heuristic method. This method sorts the orders according to a function based on profit, penalty, and due date. The orders are then produced in the sorted order. However, this method performed quite badly. We also tried a method based on integer programming. This method generated a very profitable schedule, but required more than the 15 seconds available per day to run. We could make it run faster, but only by limiting the horizon, which caused it to be no more effective than the heuristic method.

3.1 Greedy Algorithm

Instead we decided on a greedy scheduler, which based production on earliest due-date first. This scheduler is very fast and did a better job than both the previous ones.

One of the problems that turned up in the qualifying rounds was that once the agent had a sufficient number of late orders, a cascading effect caused all future orders late. This happened when there were enough late orders to use up most of the factory capacity for a day or more, causing the orders that had to be produced today to be produced late. These late orders, in turn, caused the following day's orders to be late. It turns out that in many cases it is better to never get around to building late orders than to let them negatively impact the schedule.

To avoid this problem, we enhanced the greedy algorithm to favor on-time orders over late ones. Also, because the agent plays a conservative strategy that is designed to avoid bidding on more orders than it can handle, the agent rarely needs to choose which orders to produce on any given day — if an order needs to be scheduled for production today, it can be. With such a bidding strategy, this simple greedy scheduler is all that is needed.

On day d, the greedy algorithm computes tomorrow's factory schedule as follows:

(1) Sort all orders on their due date, setting aside orders that are late.
(2) For orders with the same due date, sort on profit and current inventory.
(3) Until the schedule is full, schedule orders that need to be built on day $d+1$ in order to be shipped on day $d+2$. This is the latest they can be produced and be shipped on time.
(4) If there is extra capacity, schedule orders that are already late.
(5) If there is still extra capacity, schedule orders due after $d+2$.

This algorithm biases the agent in such a way that it builds order that must be produced today to avoid being late, then yesterday's orders, and then future orders.

3.2 Shipping

There is no limit on how many computers can be shipped on one day. This makes it fairly easy to decide which orders to ship. Since there is no reward for shipping an order early and no costs associated with storage, the agent might as well wait to deliver a completed order until its due date. That way, if it is possible to use the completed units for another earlier order and still be able to build more to make up for the loss, the agent will do so. Thus the strategy is to keep all completed orders in inventory until the day before their due date, and then ship as much as possible. If there are completed PCs in inventory after that, ship any overdue orders.

At the end of the game this is done slightly differently. The customer requests PCs with due dates after the game is over. The agent can then choose to either ignore these extra orders, or bid on them and make sure they are produced and shipped before the last day, day 219. Because these orders have to be produced early, and some agents decided to ignore them, there was less competition for these last orders, which often led to higher profit margins in the end game.

4. BIDDING

Bidding is an important part of the game. An agent that does not have a good bidding strategy is likely to find itself either not winning as many orders as it can handle, or bidding lower than necessary, and maybe also winning too many orders. All of these scenarios will lead to a lower score than if the agent extracted the maximum profit from just enough orders to fill its factory. A scheme where an agent bids high and wins few orders may work in games with very high demand. However, in games with average or low demand, which the game is biased toward [Estelle et al. 2003], it is very likely that the other agents will bid lower on enough of orders to make this scheme uncompetitive.

4.1 Planning Horizon

Our bidding strategy is based on the assumption that we will have ten opportunities to bid on orders that are due on day d. Thus, we don't really expect to win all of the orders, but we bid cautiously so that if all of our bids win, we will be able to satisfy the demand. The agent keeps track of its available capacity for each day via the planning horizon. The available capacity on a day limited the number of orders bid on. Not all bids will win, but because the planning horizon was far enough in the future—10 days—each day would be considered many times, and eventually fill up.

How quickly the free capacity for a day fills up is an indication of whether we are getting as much profit as we can. We want the free capacity for day d' to be consumed by the time d' is the day after tomorrow. If it is not filled up in the end, factory capacity is wasted; if it fills up too quickly, we are probably not asking for as much profit as the market is permitting. To control this, the agent maintains a *profit margin*, which indicates how much profit the agent requests on all orders. The profit margin was adjusted if the agent won too many or too few orders. This was the only adaptive part of the agent.

4.2 Pricing

As part of the conservative strategy, the supplies and capacity needed for any order on which the agent bid are reserved so the agent does not expect to be able to use them for other orders. If an order is not won, the supplies and capacity reserved for it are returned to the available pool to be put toward another order the next day. The agent uses a ranking function that sorts the orders with priority given to orders using parts with high inventory levels. We could have ranked orders based on other aspects also, like penalty, due date, etc, but we expected other agents to follow that strategy which would lead to more competition on the attractive orders. The price for an order was calculated with upper and lower bounds as follows:

$$\begin{aligned}
\text{price} &= 5 + \text{profit margin} * (\text{reserve price} - \text{our cost}) * \\
&\quad (\text{num orders} - 80)/240 \\
\text{upper bound} &= \text{reserve price} - 1 \\
\text{soft lower bound} &= \text{previous day prices} + \text{random}(30) - 10 \\
\text{hard lower bound} &= \text{our cost}.
\end{aligned}$$

Where *reserve price* is the reserve price for the order as set by the customer. *Our cost* is the most recent price the agent paid for the supplies needed for this type of computer. The $(num\ orders - 80)/240$ factor is a simple way of modeling whether the customer

demand is currently high or low. In high demand situations the agent can expect to be able to extract more profit. *Previous day prices* is the calculated previous day prices for the components needed to build this type of PC (see below). The soft lower bound has an upward drifting bias to make sure that if PackaTAC placed the lowest bit on the previous day, prices would drift upwards.

The term *previous day prices* was calculated based on the high and low prices for each type of computer the previous day.

$$\begin{aligned} \text{previous day price} &= \text{low} * (1 - \text{demand}) + \text{high} * \text{demand} \\ \text{demand} &= (\text{num computers} - 800)/1600 \end{aligned}$$

Low and *high* are the lowest and the highest price, respectively, for the previous day, as reported by the server. *Num computers* is total number of computers ordered

This function was based on the assumption that if there is high customer demand there is less competition for the orders, and the agent can squeeze more profit out of the orders. On the the other hand, if there is low demand, we expect yesterday's low prices to be a better predictor of what we have to bid to win.

In addition, PackaTAC has a feature that allows it to dump products in the market at the end of the game or if it acquires an excessive amount of supply inventories. In these situations, PackaTAC will bid a fixed amount below yesterday's low PC price regardless of the relative costs of the components. In the end game, supplies are sunk costs and extracting any revenue is better than ending the game holding inventory. In the middle game, we found that if PackaTAC happened to buy expensive components early in a low-demand game, the basic pricing logic would prevent it from bidding low enough to sell products and the agent would be sidelined for most of the game. The dumping logic enables PackaTAC to escape from this situation and get back into the game.

This approach was used toward the end of the seeding rounds as well as the final rounds, and seemed to work reasonably well. After the competition, we found a bug in the dumping logic that caused the prices to be too low, sometimes even negative. Despite the bug, the unloading technique worked quite well, and the agent usually made quite a bit of money during the last few days, and rarely got sidelined while holding large amounts of inventory.

We categorize the pricing strategy as a *following* strategy. PackaTAC did not try to undercut other agents. Rather, it lets other agents set the price and PackaTAC tries to follow the changes. It only lowers the profit margin if it is not winning enough orders, often due to other agents bidding lower. It does not try to find out which orders or computers tend to be more profitable in the long run, it only considers current inventory in the bidding scheme.

5. RESULTS

The competition was played in essentially two parts. The first part consisted of a qualifying round and two seeding rounds. The second part entailed the three final rounds played during the IJCAI conference with presentation and demonstrations of the game.

5.1 Qualifying and Seeding Rounds

During the first seeding rounds all of the agents were evolving rapidly, and few conclusions can be based on these rounds. In seeding round number two most agents' behavior stabilized. Our agent finished 16th out of 18, though some serious bugs in the agent severely hampered its performance, resulting in a very low score. These bugs were eliminated at

the end of the second seeding round, and the last few games were significantly better than the previous. Because of these irksome bugs, the seeding round score probably does not show the true effectiveness of the strategy.

5.2 Finals

Because of its poor performance in the seeding rounds, in the quarter-final PackaTAC was in the same group as the top six agents from the previous round. It turned out that these six agents played strategies very similar to the high risk strategy outlined in Section 2. Thus, in very low demand games they ended up butting heads with each other and incurred very negative scores. As a result, our conservative strategy payed off, and PackaTAC had the highest score in its bracket of the quarter finals.

During the semi-final and the final round PackaTAC did not perform as well as the previous rounds. This was partly due the the problems managing back-order quantities mentioned in Section 2.2. However, the mix of agents was also an important factor in the change in performance. In particular, PackaTAC was designed to perform well against big-order agents, and was effective but not extremely competitive against more reasonable agents. PackaTAC placed third in its group in the semi-finals—which was good enough to qualify for the finals—and ended up in fifth place overall out of original twenty agents.

6. POSSIBLE IMPROVEMENTS

Due to the hurried pace of the first year of the contest some things did not get implemented or looked into that could have improved PackaTAC's performance. For instance it would be beneficial to model is the state of the suppliers. If the agent knew the suppliers' levels of production and how long delayed supplies would be, it could make better decisions about which supplier to place orders with. Another problem that would be good to resolve was described in Section 2.2, where the agent's initial large orders were delivered very late, leading to inactivity. Another idea we lacked the time to implement was to track profit per product. Currently the agent keeps track of a global profit margin for all the products combined. If this was done per product, the agent could learn to extract more profit on products where there is less competition.

7. CONCLUSION

The design approach taken in the development of PackaTAC was to solve the problems in a simple rational way. The resulting agent was a simple, heuristic agent designed to hold relatively low inventory, while at the same time not bidding on any more orders than it can fulfill with the current inventory and free factory cycles. PackaTAC had a delivery performance far better than any other agent in the final round and almost no penalties. Its follower strategy allows it to adapt to the general state as the marketplace changes over the course of a game. The key to its success was to avoid the mutually destructive behavior that other agents exhibited in low demand games, and do reasonably well in high demand games. This conservative strategy seems to be reasonably robust, considering many of the agents that played the big-order strategy to did not make it to the final round.

REFERENCES

ARUNACHALAM, R., ERIKSSON, J., FINNE, N., JANSON, S., AND SADEH, N. 2003. The tac supply chain management game. Tech. rep., Swedish Institute for Computer Science. http://www.sics.se/tac/TAC03_spec.pdf.

CHENG, S.-F., LEUNG, E., LOCHNER, K. M., O'MALLEY, K., REEVES, D. M., SCHVARTZMAN, L. J., AND WELLMAN, M. P. 2002. Walverine: A walrasian trading agent. Tech. rep., University of Michigan. http://ai.eecs.umich.edu/people/wellman/pubs/walverine02.html.

DUGUAY, F.-O., KELLER, P., AND WAHAB, M. 2003. Tac/scm '03: Redagent team. Presentation slides.

ESTELLE, J., VOROBEYCHIK, Y., WELLMAN, M. P., SINGH, S., KIEKINTVELD, C., AND SONI, V. 2003. Strategic interaction in a supply chain game. Tech. rep., University of Michigan. http://ai.eecs.umich.edu/people/wellman/pubs/dm03evwsks.html.

WELLMAN, M. P., REEVES, D. M., LOCHNER, K. M., , AND VOROBEYCHIK, Y. 2002. Price prediction in a trading agent competition. Tech. rep., University of Michigan. http://ai.eecs.umich.edu/people/wellman/pubs/ppredict02.html.