Geep: A General Equilibrium Based Price-Taking Trading Agent

Alan Oursland
University of Texas at Austin
Artificial Intelligence Laboratory
Austin, TX 78712 USA

December 4, 2003

Abstract

Geep is an agent that competes in the Supply Chain Management series of the Trading Agent Competition. Geep attempts to achieve general equilibrium on the customer side, and acts as a price taker on the supplier side. Geep uses a scheduling pipeline to manage its production capacity as effectively as possible.

1 Introduction

General equilibrium is a stable market condition in which prices are set so that supply equals demand. A price-taking buyer's demand does not change based on the price. Geep is an agent that competes in the TAC Supply Chain Management competition. Geep's strategy is based on the idea of finding a general equilibrium solution on the customer side and acting as a price taker on the supplier side. The name Geep is derived from this strategy: **GE**neral **E**quilibrium **P**rice-taker.

2 General Equilibrium

General equilibrium consists of three conditions [1]:

1. Consumption equals production (plus endowment).
2. Each consumer maximizes its utility.
3. Each producer maximizes its profits.

In SCM this can be restated as: there is no surplus at the end of game; customers prefer lower prices; and agents prefer to maximize profits.

The different customers in SCM accept offers at different prices. This range in prices can be described with in a demand curve. A specific amount of product will sell at each price point. The agents need to find the price point that maximizes their profit. If the price is set beneath this point then there is a product shortage. If the price is set above this point then there is a product surplus.

In most SCM games, demand exceeds the total agent production. Agents should try to fully utilize their production capacity and sell goods at as high a price as possible. Most of the work in this version of Geep optimizes order processing and factory utilization.

3 Geep Architecture

Geep was designed around a scheduling pipeline. Integrated with the pipeline are a customer request response mechanism, a component ordering mechanism, and a margin controller mechanism.

3.1 Pipeline Overview

The pipeline consists of a set of plans across the days. Each day plan contains factory usage; products in and out; components in and out; production schedules; and shipping schedules. The customer response mechanism can retrieve from the pipeline the number of components and the factory time available on a given day. An offer is made on a request only if it fits into the pipeline.

The current implementation of the scheduling pipeline has one day of inefficiency. Orders sit in inventory for one day between production and shipping. This is an artifact of where the in/out records for the products are kept. The "product in" records should really be on the previous day. However, this would lead to a disagreement with the inventory records of the SCM library. There was not time to remove this inefficiency before the contest, but it should be removed in a future version.

Each day plan keeps track of:

- Balance trackers for each of the components
- Balance trackers for each of the products
- The amount of unscheduled factory time for the day
- The amount of factory time being reserved for the day
- A list of orders to begin production on the day
- A list of orders to ship on the day

The balance trackers for each of the components and products keep track of:

- Quantity coming *in* that day
- Quantity going *out* that day
- Quantity *reserved* for the day
- Quantity that this day is *borrowing* from the previous day to make up for deficits in this day
- Quantity that is being *borrowed* from this day by the next day to make up for deficits in the next day

Reserved values are used during the RFQ response process. As offers are made to customers, the products, components, and factory time needed to fulfill the offer are reserved on the day. Offers are not extended if reserved resources exceed the available resources. However, the resources are not actually marked as used until the offers are accepted as orders. The reserved values are reset after all customer RFQ responses have been sent.

On any given day n, one can determine the amount of some resource r that is available using the following formula:

$$\text{Available}_{n,r} = \text{Available}_{n-1,r} + \text{In}_{n,r} - \text{Out}_{n,r} - \text{Reserved}_{n,r} + \text{Borrowing}_{n,r} - \text{Borrowed}_{n,r}$$

Suppose an RFQ due on day ten is being considered. The system knows that this order needs to ship on day nine, and needs to start production on day six. It will first look at the amount of available factory time components on day six. If the resources required for production are available, the resources will be marked as reserved and offer will be made on that RFQ.

If a customer order is placed on that offer, then the order is added to the production list on day six and to the shipping list on day nine. The production time for the order will be

subtracted from the day six available factory time. The *out* field for each of the components used in manufacturing will be incremented by the quantity manufactured for day six. The product *in* for day nine will be incremented by the amount produced, and the product *out* on day nine will be incremented by the amount to ship. If the *in* field is smaller than the *out* plus the *borrowed* field, the day plan will borrow resources from the previous day. The *borrowing* field in the current day and the *borrowed* field in the previous day are both incremented as a result. If the previous day's *in* is less than its new *out* + *borrowed* value, it will borrow resources from its previous day. This continues until the first day is reached. When the *in* field of a resource in incremented, the same number is subtracted from the *borrowing* field which then percolates through the *borrowed* fields of the previous days. Each day has an expected value on each product and component. It knows how many it is producing, using, and how many are in use by later days. Knowing this allows it to plan without exceeding the available resources.

At the beginning of each day, all of the shipping and production commands stored in the previous day plan are executed. Each of the executions is double-checked to make sure it is successful. Successful commands are deleted. Unsuccessful commands are moved to the next day. The previous day plan is removed from the schedule pipeline. The *in* quantities on the earliest day reflect the current inventories. Available resources from the previous day are added to the *in* resources of the current day to keep inventories current. When production orders could not be met, or components were not delivered on time, these inventories can become incorrect. At this time the schedule inventories are reconciled with the actual inventory values.

An example of the pipeline is shown in table 1.

1. Offer is made for 10 units of a product with assembly of 5 due on day 10.

Day	5	6	7	8	9	10
Fact. Available	500	500	500	500	500	500
Fact. Reserve	0	**50**	0	0	0	0
Product In	4	0	1	0	0	0
Product Out	0	0	0	0	0	0
Product Reserve	0	0	0	0	0	0
Product Borrowed	0	0	0	0	0	0
Product Borrowing	0	0	0	0	0	0
Component In	25	0	0	0	0	0
Comp. Out	0	0	0	0	0	0
Comp. Reserve	0	**10**	0	0	0	0
Comp. Borrowed	0	**10**	0	0	0	0
Comp. Borrowing	**10**	0	0	0	0	0

2. Offer is accepted.

Day	5	6	7	8	9	10
Fact. Available	500	**450**	500	500	500	500
Fact. Reserve	0	**0**	0	0	0	0
Product In	4	0	1	0	**10**	0
Product Out	0	0	0	0	**10**	0
Product Reserve	0	0	0	0	0	0
Product Borrowed	0	0	0	0	0	0
Product Borrowing	0	0	0	0	0	0
Comp. In	25	0	0	0	0	0
Comp. Out	0	**10**	0	0	0	0
Comp. Reserve	0	**0**	0	0	0	0
Comp. Borrowed	0	10	0	0	0	0
Comp. Borrowing	10	0	0	0	0	0

3. Next Day.

Day	**6**	**7**	**8**	**9**	**10**	**11**
Fact. Available	450	500	500	500	500	500
Fact. Reserve	0	0	0	0	0	0
Product In	**4**	1	0	10	0	0
Product Out	0	0	0	10	0	0
Product Reserve	0	0	0	0	0	0
Product Borrowed	0	0	0	0	0	0
Product Borrowing	0	0	0	0	0	0
Comp. In	**25**	0	0	0	0	0
Comp. Out	10	0	0	0	0	0
Comp. Reserve	0	0	0	0	0	0
Comp. Borrowed	0	0	0	0	0	0
Comp. Borrowing	0	0	0	0	0	0

4. Surplus offer is made for 5 units of a product due on day 9.

Day	6	7	8	9	10	11
Fact. Available	450	500	500	500	500	500
Fact. Reserve	0	0	0	0	0	0
Product In	4	1	0	10	0	0
Product Out	0	0	0	10	0	0
Product Reserve	0	0	**5**	0	0	0
Product Borrowed	**4**	**5**	0	0	0	0
Product Borrowing	0	**4**	**5**	0	0	0
Comp. In	25	0	0	0	0	0
Comp. Out	10	0	0	0	0	0
Comp. Reserve	0	0	0	0	0	0
Comp. Borrowed	0	0	0	0	0	0
Comp. Borrowing	0	0	0	0	0	0

5. Surplus offer is accepted.

Day	6	7	8	9	10	11
Fact. Available	450	500	500	500	500	500
Fact. Reserve	0	0	0	0	0	0
Product In	4	1	0	10	0	0
Product Out	0	0	**5**	10	0	0
Product Reserve	0	0	0	0	0	0
Product Borrowed	4	5	0	0	0	0
Product Borrowing	0	4	5	0	0	0
Comp. In	25	0	0	0	0	0
Comp. Out	10	0	0	0	0	0
Comp. Reserve	0	0	0	0	0	0
Comp. Borrowed	0	0	0	0	0	0
Comp. Borrowing	0	0	0	0	0	0

6. Next Day.

Day	**7**	**8**	**9**	**10**	**11**	**12**
Fact. Available	500	500	500	500	500	500
Fact. Reserve	0	0	0	0	0	0
Product In	**5**	0	10	0	0	0
Product Out	0	5	10	0	0	0
Product Reserve	0	0	0	0	0	0
Product Borrowed	5	0	0	0	0	0
Product Borrowing	**0**	5	0	0	0	0
Comp. In	**15**	0	0	0	0	0
Comp. Out	0	0	0	0	0	0
Comp. Reserve	0	0	0	0	0	0
Comp. Borrowed	0	0	0	0	0	0
Comp. Borrowing	0	0	0	0	0	0

Table 1: An Example of the Scheduling Pipeline as orders are processed.

3.2 Customer RFQ Response Overview

The customer offer mechanism keeps track of the number of offers made and the number of offers accepted. Historical records within the single game are used to calculate an expected offer acceptance rate. This expected acceptance is used to make more offers than Geep can actually handle under the belief that only a percentage of the offers will be accepted. Occasionally, the number of accepted offer exceeds the production or component capacity for a day. These orders will end up being late.

It is expected that not all customer offers made will be accepted. The actual percentage of accepted customer offers that is used to estimate future acceptance rates. When an offer is made, the acceptance percentage estimate is used to calculate the expected resources used by the offer. These expected resource values are reserved in the schedule pipeline. Every customer RFQ is considered and an offer is made as long as the expected resources are available.

If the resources are not available to manufacture an order, the available product is checked to see if the RFQ can be met with product that is in stock. Offers made against "in stock" product use the entire RFQ quantity, not the expected order quantity. This is done to prevent over sale of the surplus product inventory. Since this order cannot be met through manufacturing, over sale will result in late delivery. I assume that surplus product not sold today can always be sold tomorrow.

3.3 Margin Controller

A margin controller mechanism is used to determine the price at which to offer products. As components are delivered, the price paid for them is used to estimate the average price paid for components. These prices are used to estimate the cost of producing a product. The margin is a multiplier of the product cost at which an offer will be made.

The margin controller is a simple profit derivative following algorithm. Each day the margin is adjusted by a small amount either up or down. The profit on accepted customer offers is calculated and compared to the profit on the previous day. If the current profit is less than the previous profit, then the margin adjustment direction is reversed. Because the profit on accepted offers is highly variable, the current profit must be lower than the previous profit for several consecutive days before the trend is reversed. During this waiting period, neither the margin nor the previous profit value is updated.
The derivative following strategy was selected since it has been shown to be compatible with other agents using the same strategy [2]. Competing pricebots that were using a derivative following strategy reached collusive monopolistic prices in an artificial economy. It is hoped that a similar state might be reached with another agent during competition.

The default margin is set very high (around 3.0) and the initial expected acceptance rate extremely low (0.001) so that offers will be made on all of the customer requests at the beginning of the game, and the agent starts off with a high profit margin.

3.4 Component Request Mechanism

Each day plan knows its deficit for each component. The plans for the ten days after the current day place orders for all component in which they have deficits. This mechanism turns out to be insufficient for obtaining components. There is generally not enough time between accepting a customer order and the order due date to order and receive components. In addition, the customer offer mechanism will not make offers where there are no components available. Finally, sometimes components are delivered late. If only the exact amounts of the required components are ordered, late component shipments will cause some production to be pushed to a later day. This will cause some orders to be shipped late. If the factory capacity has already been allocated for later days a "late bubble" will travel through the pipeline until there is room in manufacturing.

Component buffer fields were added to each day plan to combat these effects. These buffer values are generated by counting the assembly time on selecting random products until predetermined factory levels are reached. The components needed to produce these values are the day's component buffer. The day plan will place RFQ's for these components until it has received them. The plan will consider components available on the previous day before ordering more components. Available components on the previous day count towards the day's buffer to prevent the mechanism from ordering too many parts. In practice, the component buffer seems to be the primary mechanism for obtaining components.

However, the component buffering method has a tendency to over order components. Daily plans do not consider future conditions. If a plan has a shortage of components, it will attempt to order those components until the shortage is met. If suppliers are running at capacity and offer the components on the following day, the day with the shortage will proceed to order them again. Even there is a large surplus of the needed components on following days, the day with the shortage will continue to order more of that component. A few guards were inserted around the component ordering and acceptance routines to help prevent large surpluses.

Geep also implements a moderate 0-day ordering strategy. It requests one hundred of each of the processors and two hundred of the other components to be delivered each day during the first 12% (36 out of 220 days) of the contest.

4 Additional Enhancements

The following enhancements work within the architecture described above in increase the number of successful orders.

4.1 Default Factory Reserve

The estimated customer offer acceptance has some unknown variance. Sometimes the customer orders exceed the agent's manufacturing capacity for the day. In a high demand environment, the factory is always running at capacity and there is no opportunity to catch-up and fill the unexpected orders. There is already a mechanism in place to reserve factory time. The first enhancement sets the factory reserve to a non-zero value. This limits the number of offers that can be made. In a high demand economy, the agent's factory will run at some percentage of the maximum. The percentage increases when extra orders are placed, but quickly returns to the low-level baseline. The default factory reserve was hand tuned while observing several games so that the largest of the production spikes almost always just less than 100% factory utilization. This value is not updated during game play.

4.2 Surplus Product Production

The unused factory production results in a decrease in completed offers in a high demand environment. This is simply due to the fact that fewer products are produced. During surplus production any unused factory capacity is used to create extra product.

The surplus mechanism counts the products requested in rejected customer queries (requests on which offers were not made). These rejection counts are used to calculate an estimate of the number of extra products Geep could sell each day if the items were on hand. The surplus mechanism produces extra product until Geep has one day's estimated demand on hand. In high demand economies, only a fraction of the daily demand can be produced and it is sold very quickly. In low demand economies, the surplus product is not allowed to grow very far beyond the actual market demand.

Having surplus product allows Geep to make offers on requests that cannot be fulfilled by production alone and reduces the number of late deliveries.

4.3 Strategic Demand Reduction

It an economy with no other competing agents, there is probably an advantage to considering some customer requests over other others. Some requests may have a higher profit margin per assembly cycle. Other requests may have a lower late penalty. An agent could maximize its profits by bidding on desirable requests first.

However, if there is more than one agent present, and everyone is pursuing the desirable requests, competition will drive prices down. Making offers on requests with too much competition is bad and will result in low margins [3]. If an agent has any preference for requests, there is a chance that another agent will have the same preference and income will drop. Geep considers each of the customer requests in random order as a form of strategic demand reduction. It is difficult to evaluate how well this strategy works without testing in the presence of other agents.

4.4 0-Day Preemption

The strategy of 0-day component ordering is well known. Geep implements a strategy to combat this called 0-day preemption [4]. Geep places very large orders (85,000 units) of each component that each supplier produces to be delivered on day 30. Geep uses its checks around component ordering to ensure that it does not accept too many components if they are offered. While this strategy limits Geep's ability to place component orders in the future, it can deny 0-day prices to the competing agents which improves the profits for everyone across the board. To limit the damage to Geep, and to encourage other agents to implement the same strategy, Geep only performs 0-day preemption 50% of the time.

It is unclear if the 0-day preemption strategy is implemented correctly. Each request of 85,000 components is sent in a single message. It is also difficult to evaluate this strategy in the absence of other agents. I depend on the results provided by Estelle et all.

4.5 End Game Strategies

Geep's architecture does not handle the end of the game very well. In particular, deliveries are often scheduled to take place after the last day. Special code was inserted to deliver all outstanding orders during the last several days of the contest. The code attempts to produce and deliver all orders without consideration to what the pipeline is doing. These special production requests occur after the pipeline order production, but before the surplus production. In test runs, this technique cut the final day late penalty from the millions to the tens of thousands.

5 Results

The component ordering system is seriously flawed. Because a day plan does not consider future conditions, the system sometimes grossly over orders components. Component inventories as high as 25,000 have been seen at the end of the game. While the additional checks reduce the chance of this happening, it still occurs sometimes. The component order mechanism also does not take into account the end of the game. When the game ends, Geep generally has at least 3,000 of each component on hand. This represents a significant amount of money that could be profit if it were not ordered.

The margin controller is also flawed. There is an interaction between the offer acceptance percentage estimate and the margin controller that causes the margin to sink to, or below, cost. Whenever the margin is raised, the actual offer acceptance drops while the estimate stays the same. The offer mechanism does not make the same number of offers but the number of accepted offers drops along with the profit. The margin controller detects this drop in profit and lowers the margin again. With the margin lower, and the pipeline partially empty from the previous reduced requests, more offers are accepted, profits go up, and the margin is dropped even lower. At some point the controller notices that

profits are decreasing and tries to raise the margin again. The cycle repeats from this point.

The pipeline and surplus mechanisms seem to be very effective at meeting large numbers of orders. In practice games the pipeline is always full and most product is sold. The effect of each of the pipeline enhancements is shown in table 2.

Game	Simple Pipeline	+ Factory Reserve	+ Surplus	+ End Game
1	91% 1993/12/64/13/77 76%	67% 1836/11/8/19 99%	94% 2813/12/9/13/4 98%	88% 2517/11/10/11/9 99%
2	92% 2013/13/28/13/16 76%	78% 2236/15/0/12/7 97%	87% 2532/11/7/12/3 98%	94% 2831/12/2/11/6 99%
3	93% 2102/12/39/17/47 68%	73% 2062/12/3/11/2 98%	94% 2711/11/9/13/4 98%	94% 2876/14/4/12/0 98%
4	75% 2054/16/7/13/4 95%	78% 2179/18/2/14/7 94%	93% 2719/11/0/12/1 99%	94% 2920/13/7/11/7 98%
5	92% 2132/12/53/16/4 71%	79% 2281/19/9/13/9 94%	85% 2401/11/2/11/4 99%	91% 2726/18/1/11/5 99%

Table 2: Order statistics in 80 day games with different features enabled. Geep vs. four dummy agents.
Statistics:

factory utilization
time / late / missed
delivery percentage

Significant improvements can be seen in late and cancelled orders when adding the default factory reserve. Producing surplus product within the factory reserve significantly improves the number of orders completed. It is not clear that the end game strategy is very useful from this data. More tests will need to be run to determine its usefulness.

6 Future Work

While the scheduling pipeline is very useful for determining what components are available for production, it does not seem to be the correct tool for placing component orders. Other mechanisms should be tried instead. The current plan is to estimate the component flow and place component orders to match it. This new strategy should also make it easier to end the simulation with a low component inventory.

There are interactions between the margin and offer acceptance rate. The current margin controller should develop a model relating the margin to the offer acceptance rate. This would allow the margin to update the acceptance rate as it changes. This should help the margin controller to converge to an optimal solution.

The profits on each day are highly variable. A mechanism that attempts to make the same number of offers on each day might help smooth the daily profits which would allow the margin controller to be more accurate.

Observation of the margin controller makes it apparent that it is best to make offers on all customer requests to find the requests that are willing to pay the most. This suggests another approach to margin control. The margin should be adjusted to control the acceptance rate on offers across all requests.

The estimations of various values could be improved. Most of the estimates are currently use low-pass filters. Some of these, especially the offer acceptance rate, would probably be better estimated using Kalman filters. The variance on the estimated value in a Kalman filter could be used to optimize other parts of the system. For example, the default factory reserve could be adjusted to dynamically control the percentage of orders that are expected to be late.

There are several other small optimizations that could be made throughout the program. One example is sorting late orders by their penalty and attempting to fill the orders with the largest penalty. It is this author's opinion that there are many small and simple changes like this that can make the system operate more efficiently although they are not all readily apparent.

The game theoretic approaches of strategic demand reduction and 0-day preemption need to be analyzed to see if they actually have an effect.

7 Conclusion

A good general equilibrium solution in SCM requires the ability to produce and sell as much product as possible. The pipeline architecture is an effective method of fulfilling large numbers of orders. With a better component ordering system and margin controller, I believe Geep will be very competitive.

References:

1. Tuomas Sandholm. In the textbook Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence, Weiss, G., ed., MIT Press. p. 201-258.
2. Amy R. Greenwald and Jeffrey O. Kephart. Shopbots and Pricebots. In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, pages 506-511, Stockholm, Sweden, August 1999.
3. Paul S.A. Reitsma, Peter Stone, Ja'nos A. Csirik, and Michael L. Littman. Self-Enforcing Strategic Demand Reduction. In Agent-Mediated Electronic Commerce IV. Designing Mechanisms and Systems: AAMAS2002 Workshop on Agent-Mediated Electronic Commerce Bologna, Italy, July 16, 2002. Revised Papers. Pages 289-306.
4. J Estelle, Y Vorobeychik, MP Wellman, S Singh, C Kiekintveld, and V Soni. Strategic Interactions in a Supply Chain Game.