Socrates: A Production-Driven SCM Agent

Carlos R. Jaimez González and Maria Fasli

University of Essex, Department of Computer Science
Wivenhoe Park, Colchester CO4 3SQ, UK
{crjaim, mfasli}@essex.ac.uk

Abstract. The Trading Agent Competition (TAC) is an open-invitation forum designed to encourage research into electronic markets and trading agents. In this paper we present the Socrates trading agent and the strategies that were developed for and used in the TAC Supply Chain Management game as part of the 2004 competition. The resulting behaviour and performance in the TAC competition as well as in a series of controlled experiments are discussed.

1 Introduction

In today's highly interconnected and networked world more and more businesses and organizations choose to do business online. This is a dynamic environment where manufacturers may negotiate with suppliers on the one hand, while at the same time compete for customer orders and have to arrange their production schedule and delivery so that orders are delivered on time. The ability to respond to changes as they happen and adapt to variations in customer demand and the restrictions as imposed by procurement, is of paramount importance. This is the kind of environment that agent technology is best suited for: dynamic, constrained and real-time. However, to be able to build agents that offer solutions to problems such as supply chain management, we need to have a very good understanding of the domain itself and the problems that arise in it. In particular, firstly we need to gain a better understanding of the problems that arise in supply chain negotiation situations. Secondly, we need to explore strategies for coping in dynamic and competitive environments, and finally develop agent-based systems for an automated supply chain process. Trial and error in a real environment carries very high risks. To demonstrate the potential of applying agent technology in complex domains like supply chain management, realistic testbeds are required that allow researchers and practitioners to test out and evaluate ideas and techniques. The TAC Supply Chain Management (SCM) game was designed to capture many of the dynamics of such an environment and provides an ideal forum for researchers to test, evaluate and learn.

This paper presents the Socrates trading agent and the strategies that were used in the TAC Supply Chain Management game in the 2004 competition. As a good strategy for obtaining components was essential for a good overall performance in the game, we decided to concentrate on this aspect of the agent. Socrates is a production-driven agent that attempts to keep factory utilisation to the maximum for as long as possible during the game. We present the problems that arise in dealing with the suppliers and the strategies that were developed to tackle them. The rest of the paper is organized as follows. The following section presents the TAC SCM game and a section describing related work follows. Next, the Socrates trading agent and the strategies that were developed to deal in particular with the supplier problem are described. Following this, we present the results of the competition as well as those of a series of controlled experiments. The paper closes with the conclusions and pointers to future work.

2 The SCM Game

In the TAC SCM game six agents compete against each other in a 220-day game which lasts 55 minutes in real time [1]. Each agent is a manufacturer which assembles PCs from CPUs, motherboards, memories, and hard disks. CPUs and motherboards are available in two different product families, Pintel and IMD. A Pintel CPU only works with a Pintel motherboard while an IMD CPU can be incorporated only in an IMD motherboard. CPUs are available in two speeds, 2.0 and 5.0 GHz, memories in sizes, 1 and 2 GB, and disks in sizes 300 and 500 GB. The ten different components can be combined into sixteen different PC models. The agents need to procure these components from eight different suppliers. On the other side, the agents need to secure customer orders each day. Once an agent has received an order it needs to assemble the required number and model of PCs in its factory and subsequently ship the finished products. All agents start the game with no money in their bank accounts, no components or assembled PCs in their inventory, and no pending customer orders while they enjoy unlimited credit from the bank. Each model of PC requires a different number of cycles to be produced and an agent has a limited assembly capacity every day which is 2000 cycles. The TAC server simulates the suppliers, customers, and the bank and provides production, and warehousing services to the individual agents. The agent who makes the most profit at the end is declared the winner.

During the game each agent negotiates contracts with the suppliers by first sending RFQ requests for a specific type of component. Suppliers, which are utility maximisers, reply with offers to these RFQs. In particular, in response to an agent's RFQ, they may send a complete offer which matches the agent's request in terms of delivery date and quantity, or partial offers that may not match the quantity or delivery date requested. If the supplier cannot satisfy an order, no offer is sent. For every offer sent, the agent needs to determine whether to accept it and place an order, or reject it. However, if both partial and complete offers have been sent, the agent can only accept one of them.

The customers request PCs models for a specific due date indicating their reservation price by sending RFQs to all agents. The agent may reply with an offer bid, and the customer awards the order to the lowest bid. The winning agent must deliver the PCs ordered by the due date, otherwise it incurs a penalty. An agent is responsible for its production and every day needs to send a schedule indicating how many PCs and of which model will be manufactured the next day based on its assembly capacity. Obviously, for manufacturing to take place the agent needs to have the necessary components in its inventory. Components that arrive the same day can only be used in production the day after. Each day the agent also needs to prepare a delivery schedule for the next day.

The TAC SCM game features two main interrelated problems:

The supplier problem. One of the main problems in the TAC SCM game is to plan a good strategy for ordering components from the suppliers. There are many factors to take into account such as the quantity of components to be ordered, the date by which they are required, the periodicity in which they are requested, the supplier who provides them, the prices offered, the storage cost to be paid for them once they have been received as well as the possible delivery delays that may occur among other things. In essence, in dealing with the suppliers the agent acts as a reverse auctioneer in multi-attribute auctions. Issues like sole sourcing and multiple sourcing are important in this respect as well as the ability of an agent to switch from one supplier to another if needed.

The customer problem. The success of an agent does not only depend on the supplier strategy adopted, but also on dealing with the customers and selling manufactured products. An agent competes with all other agents in the game to secure customer orders. The components acquired will have been bought at different prices throughout the game so a particular PC may cost a different price say on day 80, than on day 145. On top of that, the agents have to take into consideration the cost that they have to pay while the raw components and the finished products lie in storage. This side of the SCM game resembles competing against other bidders in an auction. Although some information on aggregate price statistics is revealed during the game, this is rather limited. For a more detailed description see [3].

3 Related Work

The TAC SCM game has been running since 2003 with minor modifications every year to improve its efficacy. One of the major problems in the game is obtaining components to start production. In particular, the way that the suppliers worked in 2003, i.e. the big discount rates given for orders early in the game, gives all agents the incentive to request large quantities of components in the beginning. This problem has been coined the “0-day effect”. Most of the agents in the 2003 competition had a day-0 procurement strategy, i.e. they ordered large quantities in the beginning of the game. This was also helped by the fact that there was no storage cost for keeping massive numbers of components in the inventory.

RedAgent [11], the winner of TAC SCM’03, based its functionality on a multi-agent design, in which it used simple heuristic agents for procuring components from the suppliers. The main idea was to use internal markets to provide price estimates for the extra components that it needed to purchase. TacTex [12] used the day-0 strategy and sent extra RFQs during the game based on a prediction of the future inventory according to the current usage of components. HarTAC [7] procured components with the day-0 strategy and tried to maintain a reasonable quantity of all components in stock at all the times by ordering small quantities of components through the game; Botticelli [4] and PSUTAC [13] only relied on the day-0 strategy without sending extra RFQs during the game. DeepMaize [8] used a preemptive strategy to block agents that used day-0 strategy. The preemptive strategy worked by submitting a big RFQ to each supplier for each type of component. This RFQ had the effect of preempting subsequent RFQs because the supplier would have committed its production capacity for the rest of the game. There were few other agents using conservative strategies, such as PackaTac [6], who played a low-risk strategy maintaining a low level inventory of components. It decided to use this strategy to counteract the day-0 strategies that emerged during the competition. NaRC [5] constructed RFQs and accepted offers from suppliers based on projections for future prices and demand.

For the 2004 competition and in order to provide a disincentive for ordering and holding large numbers of components in the inventory, a storage cost was introduced and the pricing formula for the components was altered slightly. However, as it turned out this did not have a major effect on the agents' strategies regarding their RFQs to the suppliers during the first few days of the game.

4 Socrates

Socrates is an autonomous trading agent built in Java. It is production-driven, that is to say, we concentrated on strategies to deal with the suppliers, as despite the modifications made to the game, obtaining the components was essential to enable one to compete and acquire orders on the customers' side.

The development of Socrates for TAC SCM 2004 was based on the Agentware framework as provided by SICS, which includes a set of classes that provide the main functionality to establish communication with the TAC SCM server and receive all the events to be able to participate in the competition. The internal functionality of Socrates, which determines the way in which it keeps and processes information, is based on *Value Object* classes. A Value Object is a design pattern for representing objects as containers of information [9, 2]. Value Objects are used to represent transient objects to keep track of information during a TAC game. Hence, every RFQ, supplier offer, order, PC model, is represented as a Value Object. Socrates builds a number of different lists for these Value Objects in order to simplify their maintenance. Comparator classes, which are Java classes that implement the Comparable interface, are then used to sort the lists. The *Comparato*r classes are defined by specifying the property or properties of the Value Object class to be used for sorting the list. Once the lists are sorted, it is easier to manipulate them. Socrates uses a *FileManager* class which is responsible for reading the configuration files at the beginning of every TAC game. There are two configuration files, the default *aw.conf* and our own configuration file *soc.conf*. The latter one is used to specify the initial parameters to be used by the agent during the game. The information contained in the files is then passed on to Socrates. The FileManager is also responsible for storing information during the game in log files, which are then used for subsequent analysis.

4.1 Preliminaries

As we are focusing on a production-driven strategy we need to estimate the number of PCs that can be manufactured to achieve near to 100% utilisation. To determine the components to be ordered we have to consider the duration of the game (220 days), and the number of PCs that can be manufactured daily based on the agent's assembly capacity (2000 cycles). The average number of manufacturing cycles for a PC is:

$AvgCycles = sum(Cycles_i)/16 = 5.5$ (i is the PC model)

The assembly capacity for an agent during the whole game is therefore:

$TotalAssemblyCycles = Days * DayCycles = 220 * 2000 = 440000$

The number of PCs that can be assembled by an agent during the game is:
$TotalPCs = 440000/5.5 = 80000$

This calculation assumes that the agent is making full use of its assembly capacity from day 0 to 219. But this is not realistic as the first components cannot arrive before day 3 and therefore the earlier that production can start is on day 4. Moreover, production on the last day is useless as the agent is unable to deliver the PCs manufactured that day. The number of effective cycles and total number of PCs then would be:

$EffectiveCycles = 215 * 2000 = 430000$
$TotalEffectivePCs = 430000/5.5 = 78181$

We also have to take into account the assembly capacity of the suppliers in order to determine how many components a day they can produce. The expected assembly capacity of a supplier is 500 components a day, and the real production capacity for every day for each supplier is calculated as follows:

C_p(d)=max(0,C_p(d-1)+rnd(-0.05,0.05)*$C_{nominal}$+0.01*($C_{nominal}$-C_p(d-1)))

Where $C_{nominal}$ denotes the nominal or expected assembly capacity. We have determined experimentally that the average number of components that a supplier manufactures every day is 400 of each of the 2 types of components it can produce. Thus, the total number of components that can be produced during the whole game and the total number of PCs that can be manufactured can be calculated. CPUs are produced by two suppliers, Pintel and IMD, each of which provide two varieties of CPUs. If every supplier produces 400 CPUs of each variety daily, then 1600 CPUs are produced every day, which represents 1600 PCs. Considering that a supplier cannot produce any components on the first two days and that the production on the last two days is useless, we can approximate the total number of PCs that can be produced in a game:

$AllPCs = 1600 * 216 = 345600$

Assuming that all agents manufacture approximately the same number of PCs:

$345600 PCs/6\, agents = 57600\, PCs$

This number is only an approximation and can vary from game to game.

4.2 Supplier Strategies

We explored a number of strategies to deal with the suppliers in the TAC SCM 2004 competition as well as in a number of controlled experiments. The underlying strategy is based on what we call the Massive Simple Strategy (MSS) which simply sends 5 RFQs with big quantities to every supplier for every component they supply on day 0. The 5 RFQs are split taking into consideration the number of days that the components will last in the inventory, which can be computed by taking into consideration the assembly capacity of the agent and the average number of cycles needed for one PC to be manufactured. The 5 RFQs request a number of components enough to manufacture between 55000 and 65000 PCs during the entire game, which ensures a production between 150 and 180 days. For instance, the following RFQ bundle shows how Socrates splits the number of components to manufacture 56628 PCs: $\langle\langle$RFQ1,1452,10$\rangle$, $\langle$RFQ2,2178,25$\rangle$, $\langle$RFQ3,2904,49$\rangle$, $\langle$RFQ4,3630,81$\rangle$, $\langle$RFQ5,3993,121$\rangle\rangle$.

Once all these RFQs have been sent to the suppliers on day 0, the agent will receive offers for all or some of them on day 1. The basic version of the strategy considers accepting only complete or earliest complete offers whose delivery date is below a Cut-Off-Date, which is set to 180 considering that suppliers can delay the deliveries. This very simple strategy has a number of shortcomings:

1. It does not take into account that suppliers may not reply to all of the RFQs sent by the agents. If some of them are not matched by offers, the agent will end up with a number of unusable components in the inventory.
2. The same situation as above occurs if the agent receives offers with a delivery date above the Cut-Off-Date, which will not be accepted.
3. Suppliers can make offers with a delivery date below the Cut-Off-Date, but with a significant deviation from the requested date. This would cause the agent to wait for deliveries, which may lead to poor factory utilisation for a significant number of days due to lack of essential components.
4. If the components are received on time, the agent may exhaust them quite early in the game and then it remains idle.
5. The suppliers' delays may lead to gaps in the production.

To tackle the first two shortcomings, the agent detects on day 1 those RFQs that did not receive any offers from the suppliers, and also those that were not accepted by the agent because the offered delivery date was above the Cut-Off-Date. In both cases, this generates new RFQs which are re-sent to the suppliers. This process is carried out every day from day 1 onwards until the agent has received and accepted offers for all its RFQs.

Although this improvement helps, it does not solve the problem of lacking components of one or more types during a period in the game. If the agent has no memories, for instance, it cannot manufacture any PCs. This is because, although the supplier can supply the agent with the requested quantity this may be some time after all the other components essential for the production of specific types of PCs have been received. To tackle this, Socrates does not to accept earliest complete offers whose delivery date is greater than the requested date plus a fixed number of days, which was experimentally determined and set to a value between 30-40, and accepts partial offers instead (if any). The agent keeps track of the quantity of missing components of each type:

MissingComp[compId] = MissingComp[compId] + QRequested-QAccepted

Where every element in the array *MissingComp[]* is a type of component identified by *compId*; *MissingComp[compId]* is the existing quantity of missing components of type *compId* (generated due to previous partial offers accepted); *QRequested* is the quantity of components requested in the RFQ; and *QAccepted* is the quantity of components accepted in the partial offer. The missing quantities are then reordered in smaller quantities (400 components each). When suppliers send offers in response to these new RFQs, the agent accepts either partially complete or partial offers and keeps track of the number of missing components.

When components arrive on time, the agent will most likely have used its inventory for manufacturing PCs, and towards the end of the game it remains idle. To address this issue, another modification was introduced. To determine if the agent needs more components towards the end we have to ensure the agent has received all the components and calculate the number of PCs that can be manufactured with the current inventory. Considering the *LastDeliveryDate* (usually set to 121) that Socrates asks for components, the algorithm starts checking on day *LastDeliveryDate*+30 if all components have been received. If this is the case and there are enough days to manufacture more PCs, Socrates starts ordering more components in small quantities. The total quantity of components to be ordered depends on the number of remaining days in the game and the quantity of PCs that can be assembled.

4.3 Dealing with Gaps in the Production

Although the steps described so far deal with some of the problems in obtaining components and keeping the production steady, there are still gaps in the production schedule. To this end, a strategy that would detect the gaps in the factory utilisation for the whole game by analyzing all the *ActiveOrders* of components after the agent has accepted offers from suppliers for all the initial RFQs sent was deployed. The *ActiveOrders* provide information about when the components are supposed to be delivered and this will be known early in the game. The algorithm determines the gaps in the factory utilisation by looking at the delivery dates in the *ActiveOrders*:

1. Generate a virtual production of PCs with the current inventory (if any) utilising 100% of the assembly capacity, and determine the day in the game in which the production of PCs falls below 100%.
2. The day found in the previous step is used to look for *ActiveOrders* that should be delivered before that day or on that day. The agent has two alternatives: go to step 3 if there are *ActiveOrders*, or alternatively go to step 4.
3. The *ActiveOrders* found give the agent a virtual inventory of components that will be added up to the remaining inventory from step 1. The agent generates the virtual production of PCs with the new inventory (the remaining inventory from step 1 plus the virtual inventory) and determines the day in which the production stops due to lack of components. The agent continues with step 2.
4. This step is executed if there are no more *ActiveOrders*, that is the agent has no more components to manufacture for this day and probably for more days, because the next *ActiveOrders* (if any) will be delivered later in the game. This will lead to a gap of one or more days depending on when the next components arrive. The agent looks for the next *ActiveOrders* that give the 100% factory utilisation, keeps track of the days in which it cannot manufacture and goes to step 3. The number of days without production is used to generate new RFQs.
5. The algorithm attempts to fill in the detected gaps by sending RFQs only for those types of components necessary to have 100% factory utilisation during those periods in the game. The agent sends RFQs to cover production for 1 day, thus if there is a gap of 6 days in the production schedule 6 RFQs will be sent. The recursive algorithm described above is executed every TAC day until day 200 to determine if the gaps have

been covered by the RFQs sent. The agent will accept complete offers, earliest complete offers, or partial offers.

One problem with this strategy is that the virtual production can be unreliable as suppliers can delay the delivery of components. The gaps produced because of the initial RFQs can be covered, but new gaps can appear because of the delay of components.

4.4 Customers and Scheduling

Socrates' customer strategy is simple: it responds to customer RFQs which can be satisfied based on the current inventory of finished PCs. Three different ways of sorting the customer RFQs were considered and tested during the competition (a) quantity in RFQ; (b) reservation price; (c) expected profit. The method adopted was (b). Taking into account the reservation price in each RFQ, Socrates looks at the current inventory and decides which RFQs to bid for. The offered price depends on a number of factors including the current date, the quantity held in stock for the particular PC model and the average price as reported in the market report. These factors determine different levels of discounts during the game and are cumulative. However, Socrates also keeps track of how well its sales policy operates during the game by looking at the ratio of *OffersSent/OrdersReceived*. If the orders fall below certain thresholds and in combination with other conditions in the game, a price adjustment mechanism is triggered which overrides the discount price offered normally to improve Socrates' sales position. The delivery of finished PCs is arranged as soon as an order is confirmed by the customer, thus eliminating penalties. The major shortcoming of this aspect of Socrates is that future production is not taken into account.

The aim of the production schedule is to utilise the full assembly capacity of every TAC day. The agent assembles an almost equal number of PCs of each of the 16 models, provided that there are enough parts in the inventory to do so. If there is a lack of one or more components, which precludes the manufacturing of some particular PC models, Socrates adjusts its production so that an equal number of the other PC models can be manufactured. The production schedule does not take into account customer demand which is a major shortcoming. One way to address this is to either look at the customer RFQs and the most wanted or unwanted models of PCs and manufacture less of those, or look at the inventory of current PCs and if the quantities held reach certain thresholds, the production of those PC models can be temporarily suspended.

5 Results

The following supplier strategies were developed on top of the MSS strategy:

• Multi-Attempt Massive Strategy (MAMS): Tackles problems 1 and 2 by re-sending those RFQs that did not receive offers or are above the Cut-Off-Date.
• Enhanced Multi-Attempt Massive Strategy with Last Ordering (EMAMS-LO): This strategy works as MAMS, but it also addresses problem (4) by ordering components towards the end.

• Multi-Attempt Massive Strategy with Prediction of Gaps (MAMS-PG): As MAMS, but this strategy addresses problem (5) by predicting gaps in the production given the current inventory and the active orders.
• Multi-Attempt Massive Strategy with Small Orders (MAMS-SO): This strategy operates as MAMS, but orders small quantities of components for a period of time determined at the beginning of the game in an attempt to avoid gaps in the production (although these are not predicted as in the previous variation).

5.1 TAC SCM 2004

The TAC SMC 2004 competition involved five rounds: (a) qualifying, (b) seeding, (c) quarter finals, (d) semi-finals, and (e) finals. In the qualifying round 31 teams participated and games were played 24 hours each weekday for two weeks. All agents that were active over 50% of the games were allowed to advance to the seeding round (30 teams). The seeding rounds took place over a two-week period similarly to the qualifying round. The success of an agent was measured by a weighted average of its scores in all games for which it was scheduled. Games played during the second week were worth twice as much as those played in the first week. The top 24 agents proceeded to the quarter finals for which they were divided in four groups according to their final position in the seeding round:

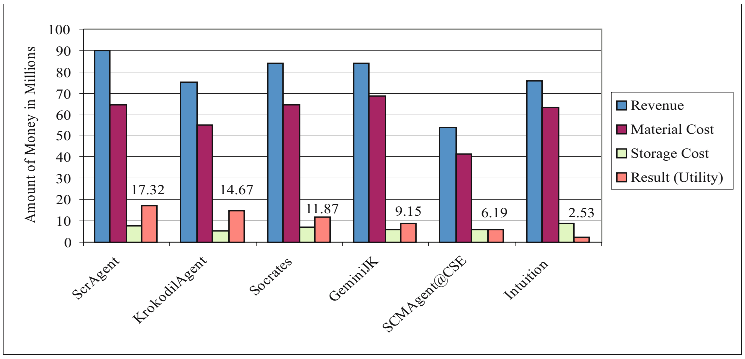
group A: positions 1, 2, 3, 22, 23, 24; group B: positions 4, 5, 6, 19, 20, 21.
group C: positions 7, 8, 9, 16, 17, 18; group D: positions 10, 11, 12, 13, 14, 15.

Every group played 8 games. The top three teams from each group progressed to the semi-finals in which the agents were divided into two groups: group 1: agents from groups A and D, and group 2: agents from groups B and C. The top three teams in each semifinal progressed to the finals, in which they played 16 games in total. The top scoring agent in the finals was declared the winner.

In the first week of the qualifying rounds the MSS strategy was used and Socrates ended up in position 20 - this was due to the problems discussed in section 4.2. In the second week of the qualifying rounds the EMAMS-LO strategy was used. EMAMS-LO was quite effective in keeping the factory utilisation up towards the end of the game, if Socrates had already received the other components on time. One disadvantage of this strategy is that in some games it is not worth ordering components towards the end as the difference between the cost of the components purchased and the price of the PCs sold gives the agent no profit. Socrates was placed 16th at the end of the 2nd week.

In the seeding round Socrates (76 games) the strategy used was MAMS-PG which attempts to predict gaps in the production after accepting all the initial offers from the suppliers. Factory utilisation improved from 75% in the qualifying rounds to 81% in the seeding rounds, and the average score improved as well. Socrates was placed in the 13th position at the end of the seeding rounds.

Socrates qualified for the quarter finals and was placed in the most competitive group, D, in which all agents seemed to have similar average scores and used similar strategies. Socrates strategy in the quarter finals was EMAMS. The performance of EMAMS was



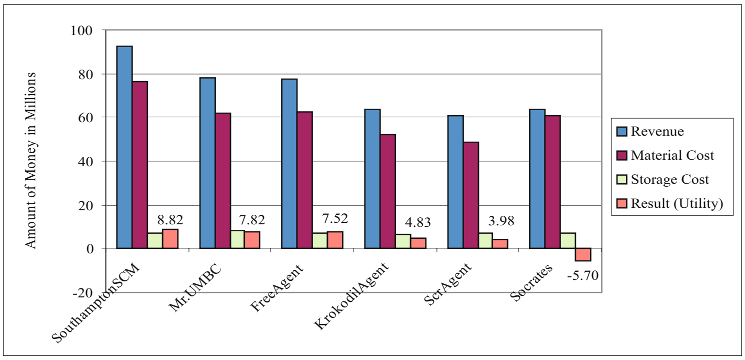
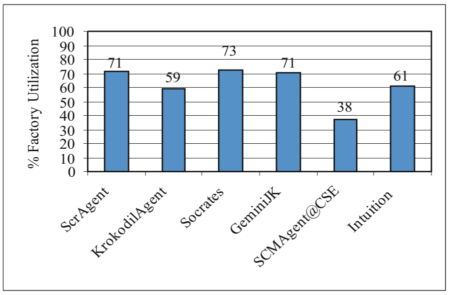
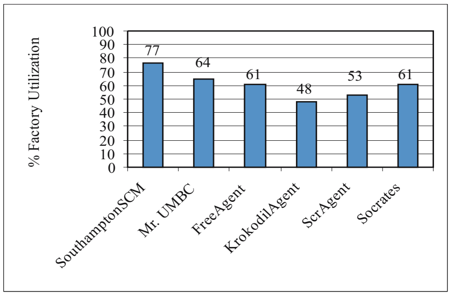


Fig. 1. Top: Average score of Socrates using EMAMS in quarter finals. Bottom: Average score of Socrates using MAMS-SO in semi-finals.



(a)



(b)

Fig. 2. Factory utilisation of Socrates with: (a) EMAMS in quarter finals; and (b) MAMS-SO in semi-finals

good in most of the games. The results from this round for group D are summarized in Fig. 1(a), 2(a) and 3(a).

Based on the quarter final results Socrates qualified for the semi-finals. The semifinal round which involved 16 games was a highly competitive environment with agents with very similar strategies procuring components on day 0. In this round, Socrates played with the MAMS-SO strategy. The strategy was not able to succeed in this

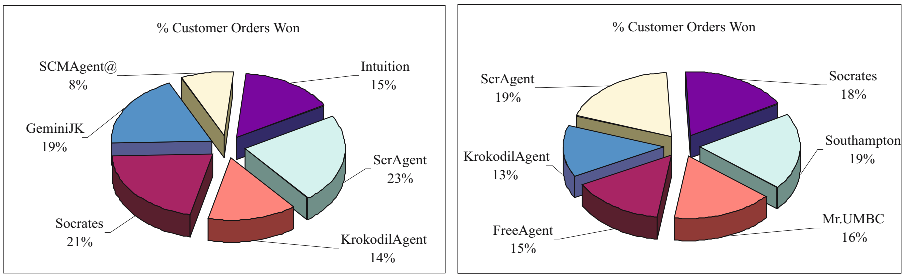


Fig. 3. Average percentage of customer orders won by each of the agents in (a) the quarter finals, (b) the semi-finals

Number of Socrates Agents

Number of M-S Agents \ Number of C-S Agents	0	1	2	3	4	5
0	6	5	4	3	2	1
1	5	4	3	2	1	
2	4	3	2	1		
3	3	2	1			
4	2	1				
5	1					

(a)

Average of Factory Utilization for Socrates Agents

Number of M-S Agents \ Number of C-S Agents	0	1	2	3	4	5
0	73	77	82	86	86	86
1	74	79	84	86	86	
2	74	81	84	86		
3	76	81	86			
4	78	82				
5	80					

(b)

Fig. 4. (a) Number of agents per game; (b) Factory utilisation of Socrates agents in respective game

environment as SouthamptonTAC was using a strategy (initially used the day before by another agent) which consisted of purchasing large quantities of memories by sending RFQs on day 0 to all the suppliers of this type of component. The agents whose RFQs were processed after the RFQs of SouthamptonTAC suffered from the lack of this type of component for long periods since all of the suppliers' production capacity was committed. As a result and since an agent is not allowed to change its strategy during a round, Socrates did not qualify for the next round. This also had an impact on the other agents. The overall results are shown in Fig. 1(b), 2(b) 3(b). The top scoring agent of TAC SCM 2004 was FreeAgent, followed by Mr.UMBC and UMTac-04.

5.2 Controlled Experiments

To evaluate the performance of our agent, in terms of factory utilisation, we ran a set of controlled experiments (in a similar way to [10]) in which we included 3 types of agents according to the way in which they order components:

• Conservative-Strategy Agents (C-S Agents) which order in small quantities.
• Massive-Strategy Agents (M-S Agents) which order large quantities in the beginning of the game to guarantee production the rest of the game.

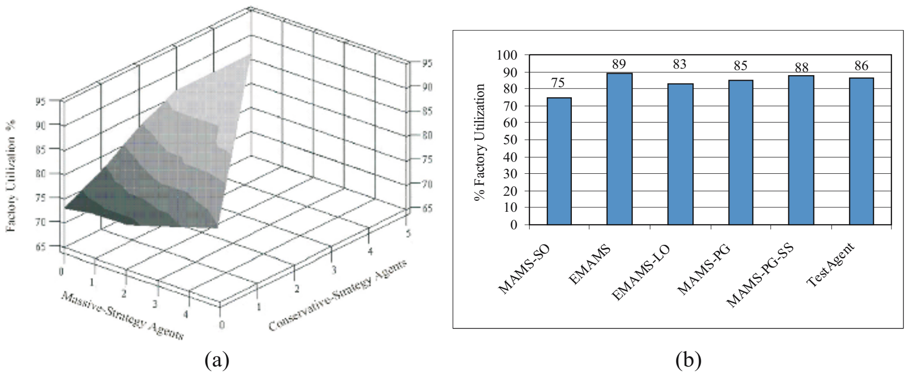


Fig. 5. (a) Average factory utilisation for Socrates agents playing against M-S and C-S agents. (b) Factory utilisation in experiments using the MAMS-PG-SS strategy.

• Socrates Agents. These use the Multi-Attempt Massive Strategy with Prediction of Gaps which orders big quantities at the beginning and fills the gaps throughout the game to maintain 100% factory utilisation (section 3).

Fig. 4(a) shows the 21 different scenarios used in the experiments, in which we have all the possible combinations of C-S, M-S and Socrates agents for a TAC 6-player game. The first line of the table indicates the number of C-S agents, the first column the number of M-S agents, while the main part of the table indicates the number of Socrates agents in the game. We ran 50 games for each scenario to test the performance of the agents in the different environments.

Fig. 4(b) shows the average of factory utilisation for Socrates agents in each scenario. For example for a game with 2 C-S agents and 1 M-S agent there are 3 Socrates agents, which achieve in average a factory utilisation of 84%. When there is more than 1 Socrates agent in a game we take the average of factory utilisation. Fig. 4(b) indicates that in non-competitive environments the factory utilisation for Socrates agents is better than that in competitive environments. We define a non-competitive environment as the environment in which the number of C-S agents is less than the sum of M-S agents and Socrates agents:

(# of C-S agents) <(# of M-S agents) + (Socrates agents)

Those environments in which the sum of M-S agents and Socrates agents is greater than the number of C-S agents are referred to as "competitive". Fig. 5(a) graphically illustrates the results from Fig. 4(a) in which as one can observe the more Socrates agents playing in a game (including M-S agents), the more competitive the environment is. The z axis in Fig 5(a) shows the average percentage of factory utilisation for Socrates agents, which demonstrates that they behave much better in non-competitive environments than they do in competitive ones.

One of the main weaknesses of the supplier strategies that were considered and tested during TAC SCM is their inability to adapt dynamically to conditions in the game that affect the procurement of components. One possibility was to allow the agent to switch between suppliers and components when one supplier fails to supply them. This new strategy, Multi-Attempt Massive Strategy with Prediction of Gaps with Supplier Switching (MAMS-PG-SS), combines the prediction of gaps with a strategy that looks for substitutable components. It determines the set of components that can be substituted for every component that the agent can order. For instance, if Socrates cannot obtain memory 1GB from supplier Queenmax, then it looks for substitutable components in the following order: memory 1 GB (MEC), memory 2 GB (Queenmax) and finally memory 2 (MEC). This list indicates the order in which the agent will try to satisfy the number of components needed. The agent keeps track of the number of attempts to every supplier and if it reaches a threshold value the agent switches supplier or in the appropriate case supplier and type of component. This process is carried out until the agent satisfies the quantity needed of every component. The strategy works as expected when a supplier cannot provide the specific component, and the agent has to look for a replacement. Since it works by substituting the supplier or the type of component, the number of PCs of each model assembled in a TAC day will be different. In some cases, the agent will be left with a number of unused components in its inventory. This occurs when the agent has ordered a greater number of CPUs than the number of motherboards of one specific type (Pintel or IMD) in its attempt to find substitutable components and manufacture alternative PC models. We run a series of 100 games to compare this strategy against all the previous ones developed, while introducing a SouthamptonTAC-like behaviour (TestAgent) to observe the results. However, as we haven't had the chance to test this strategy against other real agents and all our strategies operate in a similar way, the results as shown in Fig. 5(b) are not conclusive.

6 Conclusions

This paper presented the Socrates trading agent and the strategies employed in the TAC SCM 2004 competition. Socrates is a production-driven agent which attempts to keep factory utilisation to the maximum throughout the game. As such it is plagued by a number of problems as has been described, more so as customer demand is not taken into consideration when making orders to the suppliers or scheduling production. We hope to address some of these problems for the TAC SCM 2006 competition. In particular, we are working on a strategy that concentrates equally on the customers' side as well as on the supplier side. We would like to explore the idea of using variable scoring functions to deal with the supplier side. We are also considering the scheduling problem under a new light, using a daily schedule as well as a dynamic global production schedule that takes into account the current inventory of components and the component *ActiveOrders* in deciding which customer orders can be satisfied.

References

1. TAC SCM specification. Available at http://www.sics.se/tac/.
2. Sun Java Center J2EE Patterns. http://java.sun.com/developer/technicalArticles/J2EE/patterns/, March 2001.

3. R. Arunachalam and N. Sadeh. The 2003 supply chain management trading agent competition. In *Proceedings of the Trading Agent Analysis and Design Workshop (TADA)*, 2004.
4. M. Benisch, A. Greenwald, I. Grypari, R. Lederman, V. Naroditskiy, , and M. Tschantz. Botticelli: A supply chain management agent designed to optimize under uncertainty. *SIGecom Exchanges*, 4:29–37, 2004.
5. S. Buffett and N. Scott. An algorithm for procurement in supply chain management. In *Proceedings of the Trading Agent Analysis and Design Workshop*, 2004.
6. E. Dahlgren and P. Wurman. Packatac: A conservative trading agent. *SIGecom Exchanges*, 4:38–45, 2004.
7. R. Dong, T. Tai, W. Yeung, and D. Parkes. Hartac - the harvard tac scm03 agent. In *Proceedings of the Trading Agent Analysis and Design Workshop (TADA)*, 2004.
8. J. Estelle, Y. Vorobeychik, M. Wellman, S. Singh, C. Kiekintveld, , and V. Soni. Strategic procurement in tac/scm: An empirical game-theoretic analysis. In *Proceedings of the Trading Agent Analysis and Design Workshop (TADA)*, 2004.
9. Helm R. Johnson R. Vlissides J. Gamma, E. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
10. M. He and R.N. Jennings. SouthamptonTAC: An adaptive autonomous trading agent. *ACM Transactions on Internet Technology*, 3:218–235, 2003.
11. P. Keller, F. Duguay, and D. Precup. RedAgent - Winner of TAC SCM 2003. *SIGecom Exchanges*, 4:1–8, 2004.
12. D. Pardoe and P. Stone. Tactex-03: A supply chain management agent. *SIGecom Exchanges*, 4:9–18, 2004.
13. S. Sun, V. Avasarala, T. Mullen, and J. Yen. Psutac: A trading agent designed from heuristics to knowledge. In *Proceedings of the Trading Agent Analysis and Design Workshop (TADA)*, 2004.