A Comparison of Multi-Agents Competing for Trading Agents Competition

DAN MANCAS[*]
STEFAN UDRISTOIU[**]
ECATERINA – IRINA MANOLE[*]
BOGDAN LAPADAT[**]
[*]Computer and Communication Engineering Department
Faculty of Automation, Computers and Electronics
University of Craiova
Bvd. Decebal, Nr. 107, 200440, Craiova, Dolj
ROMANIA
dan.mancas@ucv.ro, catya_ace@yahoo.com

[**]Software Engineering Department
Faculty of Automation, Computers and Electronics
University of Craiova
Bvd. Decebal, Nr. 107, 200440, Craiova, Dolj
ROMANIA
stefan@software.ucv.ro, lepadat_bogdan@yahoo.com

Abstract: - We present a comparative analysis for several multi-agents participating in Trading Agents Competition, Classic. The game is first partitioned into separate modules, for which distinct strategies may be developed. The strategies used are taken into consideration both individually and in relation with other agents, but also the game medium. Conclusions regarding possible improvements, better strategies and potential weaknesses are driven from each agent analysis. Alternatives like static market algorithms vs. dynamic market algorithms are considered in detail and advantages and disadvantages are discussed. Also a discussion of TAC market and possibilities for the stochastic system approximation with a deterministic one is realized.

Key-Words: multi – agent, artificial intelligence, autonomous trading agents, probabilistic market strategy, machine learning, agents.

1 Introduction

"Agents" are programs which operate autonomously in the market—sending bids, requesting quotes, accepting offers, and generally negotiating deals according to market rules. Although the agent's activity is ultimately determined by its programmers, the trading behavior is fully automated in that the humans do not intervene while the negotiation is in progress[1].

Trading agents face must play the market effectively; an agent must make real-time decisions in an uncertain and fast-changing environment, taking account of other agents doing the same. Capable agents rapidly assimilate market information from many sources, forecast future events, optimize complex offers and resource allocations, anticipate strategic interactions, and learn from experience. Successful trading agents adopt and extend state-of-the-art techniques from artificial intelligence, operations research, statistics, and other relevant fields[2].

2 Game Essentials

In the TAC shopping game, each "agent" is a travel agent, with the goal of assembling travel packages (from TACtown to Tampa, during a notional 5-day period). Each agent is acting on behalf of eight clients, who express their preferences for various aspects of the trip. The objective of the travel agent is to maximize the total satisfaction of its clients. Travel packages consist of the following:

- A round-trip flight,
- A hotel reservation, and
- Tickets to some of the following entertainment events
  - Alligator wrestling
  - Amusement park Museum.

There are obvious interdependencies, as the traveler needs a hotel for every night between arrival and departure of the flight, and can attend entertainment events only during that interval. In addition, the clients have individual preferences over which days they are in Tampa, the type of hotel, and which entertainment they want. All three types of goods (flights, hotels, entertainment) are traded in separate markets with different rules.

A run of the game is called an *instance*. Several instances of the game are played during each round of the competition in order to evaluate each agent's average performance and to smooth the variations in client preferences.

2.1 Flight tickets

In TAC Classic game, TACAIR is the only airline. Flight tickets are sold in single seller auction, and each agent can only buy it. Tickets for these flights are sold in single seller auctions---one auction for each day and direction (in or out). Since all clients must stay at least one night in Tampa, there will be no inflights on the last day, nor outflights on the first day. The auctions will clear continuously. TACAIR is represented in the marketplace by an agent that sets prices according to a stochastic function. Each ticket has an individually appointed date during the first day to the fifth day and direction (inflight,outflight). Since all clients must stay at least one night, there are no inflights on the last day, nor outflights on the first day. The price of a flight ticket is updated at random every 10 seconds according to the stochastic function decided in advance. A flight ticket is not sold out, and if an agent bids at a price higher than a current ask price, it can purchase immediately at the ask price[4].

2.2 Hotels

There are two hotels: The Tampa Towers(TT) and The Shoreline Shanties(SS). TT is a hotel better than SS, so we would expect to cost more. A client can not move between hotels. Each hotel has 16 rooms per day during the first day to the fourth day. There is one auction every combination of hotel and night, each with 16 rooms. So, there are eight auctions in total. They are traded in the ascending multi-unit auction. These auctions are closed one by one randomly during four minute to eleven minute every minute, and the auction clears when it closes. The ask price is sixteenth highest price. The bid submitted once cannot be canceled. And when agent submits new bid, they must follow the following rules. Since clients need hotels only from the night of their arrival and through the night before their departure, no hotels will be available (or needed) on the last day[4].

2.3 Entertainment tickets

At commencement of a game instance, each travel agent receives an allotment of entertainment tickets. There are three kinds of entertainment tickets:

- Alligator wrestling,
- Amusement park,
- Museum.

There are a total of 8 tickets available for each event type on each day, and each agent receives 12 tickets partitioned as follows:

- One bundle of four of a particular type on day 1 or day 4.
- One bundle of four of a particular type on day 2 or day 3.
- One bundle of two of a particular type (different from above) on day 1 or day 4.
- One bundle of two of a particular type (different from above) on day 2 or day 3.

Each ticket has an individually appointed date during the first day to the fourth day, and it cannot be used for other day. A client cannot use a ticket on the day of departure. At the time of a game start, each agent receives the fixed number of tickets randomly. These tickets are traded in continuous double auction, and agent can buy and sell. A price is determined according to the balance of supply and demand. Note that only one kind of ticket can be used for one day. An agent trades in these goods and assembles a travel package. Each client has individual preferences, which are a preferred arrival date (PA), a preferred departure date (PD), a bonus of staying in TT (HB), and a bonus of every kind entertainment ticket (AW;AP;MU). And each client's utility (U) is calculated by the following formulas according to these preferences[4].

$$U = 1000 - travel\_penalty + hotel\_bonus + \\ + entertainment\_bonus \quad (1)$$

$$travvel\_penalty = (|AA - PA| + |AD - PD|) \quad (2)$$

$$hotel\_bonus = TT \times HP \quad (3)$$

$$ent\_bonus = AW\,?\times AW + AP\,?\times AP + MU\,?\times MU \quad (4)$$

where $AW\,?, AP\,?, MU\,?$ each in $\{0,1\}$ are ticket indicators for each event type.

2.4 Bid auctions

All of the auctions run according to the following high-level protocol:

1. An agent submits a bid to the auction.
2. The auction updates its price quote, indicating the current going prices.

A bid contains a bid string, representing an agent's willingness to buy and sell the good in an auction. A bid string containing a list of bid points in the following form:

"($(q_1\ p_1)$ $(q_2\ p_2)$ ... $(q_n\ p_n)$)" (5)

where q_i is a quantity and p_i is a price. If there is a point $(q_i\ p_i)$ with $q_i > 0$, then it means that the agent is willing to buy q_i units of the good at the auction for no more than p_i price units per unit of the good. If there is a pair $(q_j\ p_j)$ with $q_j < 0$, then it means that the agent is willing to sell qj units of the good at the auction for no less than p_j price units per unit of the good. The prices should always be nonnegative[4].

3 Means of Comparing the Agents

We will base our comparison for the strategies adopted by various agents on three important criteria, deriving from the game itself. In fact, besides the idea that without a flight ticket there can be no vacation, and without a hotel the package is not feasible, between these three components there seems it doesn't exist a certain interdependency. Basically, if for all of the above, the agents' strategy would be to get the lowest price possible, we could separate these three components as distinct pieces of the overall architecture of an agent. Therefore we will look at strategies concerning:

- Flight Booking,
- Hotel Booking
- Entertainment Tickets Buying.

We will take into consideration three agents that took part in the TAC Classic competition on which we will perform the analysis: Mertacor, Walverine, Roxy-Bot. All of these three agents were in the TAC finals, in the previous years [5][6][7][8].

3.1 Flight booking

Flight auctions are continuous one-sided auctions, and close at the end of the game.
Agents may submit buy bids, but not sell bids. Only the TACAIR seller may submit sell bids.
Price quotes are issued immediately in response to new bids. The price quote is specified as the ask price, which is simply the price of the current sell bid[4].

From all the strategies, Mertacor seems to have the most elaborated one. Their approach is to predict the moment when x(t), specified in the game as the perturbation function, changes sign, as this would be a very good moment to book a flight. A first observation that should be made is that all of the bookings occur at specified time intervals, therefore there results an array of update prices depending on the moment and the equation provided by the game can be rewritten in the following manner:

$$x_i = 10 + i \cdot \frac{y}{N}, i \in [1, N] \quad (6)$$

In this equation *y=x-10*, where x is the random variable specified by the game and *N* is the total number of auctions, in our case maximum 54, due to time limitation of the game.
The Mertacor authors consider that *corner ic* of a flight auction is the latest update for which we know that price perturbations are drawn from [-10, *xi*]. Also *low bound B* is the maximum real number for which we know that $y \geq B$. They define the following equation

$$B_i = (\delta p_i - 10) \cdot \frac{N}{i} \quad (7)$$

For this it is true that $B_i \leq y$. When $B_i \geq B$ then a new low bound has been encountered. They finally estimate the corner with the equation:

$$i_c = \left\lfloor 10 \cdot \frac{N}{|B|} \right\rfloor \quad \text{if B<-10 and N otherwise} \quad (8)$$

Due to the fact that this is only an estimation of the corner and not the real corner itself, by using this strategy, they can compute only when the real corner has been crossed so that they can make a bid. Based on the evidence vector, containing the bounds and the times at which these bounds were specified they estimate y as being:

$$p(s \in S_1 \mid BE)_c = \frac{\int_{S_1} g(BE,s)}{\int_{S_2} g(BE,s)} \quad (9)$$

Where *s=y/20*, BE is the evidence bound vector, *g* is a function depending upon 2 variables which represents a probability and S_1 =[*bn*, *z*]. The authors fix the probability of equation 4 to 0.8 and derive an interval where y might be situated.

Based upon these predictions Mertacor takes its decisions. However, this is not an optimum strategy, and most of the time, it helps predict a price situated somewhere in the middle of all the flight prices, proposed during the auction. Furthermore, the estimation of the corner often falls in the second part of the auction, where the prices are known to be usually higher. Due to the fact that in this mathematical conceived model they can detect a single corner, despite the fact that there might be as many as 27, we consider this a rough estimation. A better approach would be to consider a set of corners, and try to estimate these corners. If we were to make a simple analysis of this algorithm we would deduce that it is possible in practice for this algorithm to tolerate a price rise of 102 after 25 bidding periods. More than that, if an important increase of the price would be detected, the agent would bid immediately. Therefore, the agent will never bid on the smallest price, and there is almost no chance for it to do so, unless a sufficient increase would follow, the price bidding price will not be estimated correctly, and then a drastic decrease would happen. As it can be seen this is a highly improbable case. Therefore we can conclude that this agent will not buy conveniently in most of the cases.
More than that, the estimation of the corner is never optimum. The corner in conformity to equation (3) can be found after the middle of the auction. Other authors consider that the price for flight auctions have the tendency to rise most of the time, so bidding after the middle of the auction might result in disastrous consequences.
Walverine maintains a distribution Pr(x) for each flight, initialized to be uniformon [-10,30], and updated using Bayes's rule given the observed perturbations at each iteration:

$$Pr(x|\Delta) = \alpha\, Pr(x)\, Pr(\Delta|x) \quad (10)$$

Given this distribution over the hidden x parameter, the expected perturbation for the next iteration, $E[\Delta' \mid x]$, is simply $(lb+ub)/2$. Averaging over the distribution for x, it is obtained

$$E[\Delta'] = \sum_x Pr(x)\, E[\Delta'] \quad (11)$$

Given a set of flights that Walverine has calculated to be in the optimal package, it decides which to purchase now as a function of the expected perturbations, current holdings, and marginal flight values. On a high level, the strategy is designed to defer purchase of flights that are not quickly increasing, allowing for flexibility in avoiding expensive hotels as hotel price information is revealed.
RoxyBot proposes almost the same strategy as Wolverine. It holds some point estimates that it uses during scenarios.

3.2 Hotel booking

Hotel auctions are standard English ascending multi-unit, except that they close at randomly determined times. Specifically, at 01:00 (one minute) into the game, one randomly chosen hotel auction will close. Another hotel auction closes each minute thereafter, on the minute, again chosen randomly, until 8:00 when the last hotel auction is closed. The agents cannot tell in advance which hotel auction will close at which time. A hotel auction clears and matches bids only once, when it closes. Price quotes are only generated once per minute, on the minute.
Agents may submit buy bids, but not sell bids. Only the hotel owners may submit sell bids. The hotel owners submit bids to provide up to 16 rooms of each hotel type on each night, for a minimum price of $0[4].

Pertinent observations were made when designing

Mertacor:

$$s_{in,i} = 0.5 - 0.1i, i \in [1,4] \quad (12)$$

$$s_{out,j} = 0.1j - 0.1, i \in [2,5] \quad (13)$$

The above probabilities represent the probability that an inflight might occur on the i-th day, and the

probability that an outflight might occur on the j-th day respectively. This should be taken into account by all agents, when designing a strategy for this point. Unfortunately only Mertacor does this, the rest of the agents use other approaches. Mertacor approach is to compute estimations of the hotel requests depending on days.

$$\varepsilon i = Ii / I$$
$$\mu i = Oj / O,$$

$$i \in [1,4],\ j \in [2,5]. \quad (14)$$

The equations above represent the *relative* inflight and out-flight ticket demand.

$$H1 = I1$$
$$H2 = I1 + I2 - O2$$
$$H3 = -I4 + O4 + O5$$
$$H4 = O5 \quad (15)$$

H represents absolute hotel room demand. Thus equation (8) can be rewritten:

$$H = R \cdot F \quad (16)$$

where

$$H = (H1\ H2\ H3\ H4)^T$$

$$R = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$F = (I1\ I2\ I3\ I4\ O2\ O3\ O4\ O5)^T$$

The estimated hotel demand can be computed using the formula:

$$\hat{H} = R \cdot \hat{F} \quad (17)$$

In this case contains relative estimations of the hotel requests. This strategy has a very important problem. Mertacor acts as a single agent, on a single market, without taking into consideration the other agents participating in the competition. With concern to Hotel bids the price is determined by the market, so probabilistic computations might not hold the required result all the time. However Mertacor, uses previous experience of the games and adapts its biddings baring in mind the previous results. Also this kind of analysis works sometimes, but sometimes might result in some less fortunate decisions.

This situation would require a strategy that is adapted to market conditions. This kind of economical approach is used by Walverine. Walverine predicts hotel prices based on a literal application of its presumption that TAC markets are competitive. Specifically, it calculates the *Walrasian competitive equilibrium* of the TAC economy, defined as the set of prices at which all markets would clear, assuming all agents behave as price takers, i.e., behave competitively.

Demand for a given hotel is a function of *all* hotel prices, as changing the price of any hotel can affect the agent's choice of trips, and thus the demand for any other hotel.

The interconnection of markets renders this a problem in *general equilibrium* (as opposed to partial equilibrium), and prevents us from analyzing each hotel in isolation.

Prices constitute a *competitive equilibrium* if aggregate demand equals aggregate supply for all hotels. Since there are 16 rooms available for each hotel on each day, we have in competitive equilibrium, *x(p)=16.*

Walverine searches for a competitive equilibrium using tatonnement. Starting from an initial guess, it iteratively computes a revised price vector according to:

$$p^{t+1} = p^t + \alpha^t (x(p^t) - 16) \quad (18)$$

Eventually the price would reach an approximate equilibrium, which should be sufficient to know what bid should the agent make next. Wolverine partitions the problem of expected demand into two components, the internal one and the external one.

$$x(p) = x_w(p) + x_{\bar{w}}(p) \quad (19)$$

Thus:

$$E[x_w(p)] = E\left[\sum_{1}^{56} x_{client}(p)\right] \quad (20)$$

This equation can be rewritten:

$$E[x_w(p)] = 56 \cdot E[x_{client}(p)] \quad (21)$$

At the beginning of the game when there are no holdings of flights and hotels, the agent optimization problem is indeed separable by client, and so (16) is justified. At interim points when agents hold goods, the demand optimization problem is no longer separable. This is actually where the problem with this strategy occurs. The authors are happy with an approximation, which is not a very good bound. Instead of using this formula, they have the possibility to combine the formula with an estimation of the goods bought by other agents. Exactly for this purpose of predicting the market and other agents' actions, Wolverine has a strategy of Hierarchical Game Reduction. Since the environment is stochastic, numerous samples (say 12) are required to produce a reliable estimate for even one profile. At roughly two hours per profile, exhaustively exploring profile space will require 13 trillion hours simply to estimate the payoff function representing the game under analysis. If the game is symmetric, we can exploit that fact to reduce the number of distinct profiles to 1, which will require 628 million hours. That is quite a bit less, but still much more time than we have. The idea of hierarchical game reduction is that although a strategy's payoff does depend on the play of other agents, it may be relatively insensitive to the exact numbers of other agents playing particular strategies.
For example, let (s,k;s') denote a profile where k other agents play strategy s, and the rest play s'. In many natural games, the payoff for the respective strategies in this profile will vary smoothly with k. If such is the case, we sacrifice relatively little fidelity by restricting attention to subsets of profiles, for instance those with only even numbers of any particular strategy. To do so essentially transforms the N-player game to an N/2-player game over the same strategy set, where the payoffs to a profile in the reduced game are simply those from the original game where each strategy in the reduced profile is played twice.

The potential savings from reduced games are considerable, as they contain combinatorially fewer profiles.
Lastly they compute the excess for each client:

$$E[x_{client}(p)] = 0{,}1 \cdot E[\sum_{(pa,pd)} E(x_{pa,pd}(p)] \qquad (22)$$

Based upon these formulas Walverine obtains the necessary estimations. The policy proposed is tested on real markets, and it seems that it also giving results in TAC according to the results Wolverine had. The strategy Wolverine uses, combined with the technique of predicting adversary strategies holds good results. However, some problems still arise from the fact that the decisions are taken based upon only two other strategies.

Roxy-Bot also uses Walverine approach. The hotel price predictions are evaluated using two metrics: Euclidean distance and "expected value of perfect prediction" (EVPP). Euclidean distance is a standard way of measuring the difference between two vectors, in this case the actual and the predicted prices. The value of perfect prediction (VPP) for a client is the difference between the value of the best package for the client based on the actual prices and the value of the best package for the client based on the predicted prices. EVPP is the expected VPP averaged over the client distribution.
They interpret each prediction with 56 random clients as a sample scenario, so that a set of such scenarios represents draws from a probability distribution over competitive equilibrium prices. The vector of predicted prices that is evaluated and plotted is the average of multiple (40) such predictions. Using random clients helps us make better interim predictions later in the game as we explain next.
As hotel auctions close, RoxyBot-06 updates the predicted clearing prices of the open hotel auctions. The first is to fix the prices of the closed auctions at their clearing prices and then to run SimAA or tatonnement with expected or random clients. The second is to distribute goods from the closed auctions to the clients who want them the most, and then to exclude any closed auctions in further runs of SimAA or tatonnement.

Note that we can only distribute goods to random clients. It is not clear how to distribute goods to "expected clients," which are aggregate clients rather than real clients. Figure 1 (center and right) shows that the predictions based on the distribution method are better than the others. Hotels that close early tend to sell for less than hotels that close late; hence, any method that makes relatively constant predictions all throughout the game is bound to suffer.

There can be observed that the Roxy-Bot team has observed well the minuses that Wolverine was exhibiting and corrected them.

3.3 Entertainment booking

The entertainment ticket auctions are standard continuous double auctions (much like a stock

market) that close when the game ends. Agents may submit bids with buy and/or sell points (so long as a bid does not specify that the agent sell to itself).

Entertainment ticket auctions clear continuously.
Bids match immediately, if possible. A bid that does not completely match remains standing in the auction.
Buy bid points will immediately match the lowest price standing sell bid points that have prices at or below the price of the buy bid. Sell bid points will immediately match the highest price standing buy bid points that have prices above the price at or below the sell bid. Bids match at the price of the standing bid in the auction.

Regarding Mertacor strategy for Entertainment Selling and Purchasing, they use the following algorithm:

```
PROCEDURE MertacorSellStrategy
1:    FOR each entertainment ticket in possession
2:        Assign a pre-specified value to target;
3:        mean ← getMeanValue();
4:        M ← A*target + B*mean;
5:        IF M ≤ (1/2)*target OR M ≥ (3/2)*target
THEN
6:            M ← relocateM();
7:        END IF
8:        V ← calcVal();
9:        IF V < Vo THEN V ← Vo; END IF
10:       profit ← ask– 2*V;
11:       Mt ← w(t)*M;
12:       Ra ← M;
13:       Rb ← 3*M/2;
14:       Rc ← rand(M/2, M);
15:       IF profit ≥ Mt-Ra THEN sellTicket();
16:       ELSE IF profit ≥ Mt-Rb THEN ask (Mt-
              Ra+2*V);
17:       ELSE ask (Mt - Rc +2*V);
18:       END IF
19:   END FOR
```

Mertacor uses previous game experience and prespecifies values to some parameters inside its strategy. Target gets values from interval [5,12]. This assignment of the variable was deduced through experiments. Other parameters deduced by experiment are A and B. Mean is computed from previous games. M is then bounded between *(1/2)*target and (3/2)*target*. The reason is that Mertacor designers want to make sure that this value would allow obtaining a good price for buying, or a credible price for selling. A value V for the price is then computed and if that particular value is considered to be less than a minimum value, the minimum value becomes the desired V value. A profit is afterwards computed as being the ask price from which there is subtracted the value of the ticket multiplied by 2. Then parameters *Mt, Ra, Rb, Rc* are computed and a selling decision or a buying decision is made afterwards.
The strategy Mertacor uses has some disadvantages. Firstly, Mertacor does not make a computation regarding the utility it will get by using the ticket itself instead of just selling it or buying it. The decision is made for a static market, one that does not change prices much. This could make Mertacor an isolated agent on the market and serious problems might occur from this.
Furthermore, w(t) function is a lineary and continous one, and therefore a predictor for the other agents might be possible, in order to know what prices might they get for tickets, when to buy them, what bidding price should they have to increase their utility. Lastly, Mertacor does not take its decisions based upon market reality he observes, but rather he uses precomputed parameters from other games.
Walverine adopts a fairly minimal adjustment of its basic (initial) price prediction method to address ticket bidding. In calculating its own demand for tickets, it takes into account its current holdings of flights and closed hotels. For closed hotels, Walverine fixes its own demand at actual holdings. For other agents, it continues to employ initial flight prices in best-trip calculations. Since they do not know the holdings of other agents, they make no attempt to account for this in estimating their demand. This applies even to closed hotels—in the absence of information about their allocation, Walverine's tatonnement calculations attempt to balance supply and demand for these as well.
Also Walverine computes the entertainment expected surplus. As a result taking into account previous games they obtain the following table:

TABLE 1. Walverine's game results

	Expected Entertainment Surplus	
Arrive:Depart	TAC-01 Prices	TAC-02 Prices
1:2,4:5	74.7	78.0
1:3,3:5	101.5	112.1
1:4,2:5	106.9	119.9
1:5	112.7	120.9
2:3,3:4	66.2	76.6
2:4	93.0	110.7

The problem with this approach is again the fact that it is possible for other agents to come with a strategy different than what is on the market in order to destabilize it. Under these circumstances, these estimations would no longer be of use and bad decisions might be taken. In this case a strategy similar to Hotel Booking would be better, but here there are three skinds of utilities and the number of scenarios would rise in accordance more rapidly. Therefore, a combined strategy might work better in this case.
RoxyBot-06's estimates of entertainment ticket prices are based on historical data from the past 40 games. To generate a scenario, a sample game is drawn at random from this collection, and the sequences of entertainment bid, ask, and transaction prices are extracted. Given such a history, for each auction a, let trade ai denote the price at which the last trade before time i transacted; this value is initialized to 200 for buying and 0 for selling. In addition, let bid ai denote the bid price at time i, and let ask ai denote the ask price at time i.
To predict current buy price in auction a at time t, RoxyBot-06 first computes the minimum among the historical trade and ask prices at time t and the current ask price in the present game. The current buy price is then constrained to be above the current bid price in the present game. Without this latter constraint, the agent might be inclined to buy a good at a price that is lower than the outstanding bid, which is impossible.
Roxy-Bot strategy is also based on previous games. As a conclusion, in regarding the Ticket auctions, no agent from the ones that we observed is taking into account market requests and market evolution. Due to the fact that this part of the game is highly economical a good strategy would be to observe the market behavior and take decisions in accordance with that. A Walrasian approach, in relation with bounds imposed to the market would produce a far better result.
In pseudocode, the algorithm is defined as:

1: INPUT
2: current ask_est, bid_est
3: current lo_ask, hi_bid
4: rates of adjustment α, β.
5: OUTPUT :
6: adjusted ask est, bid est
7: IF (a recent trade took place at price p)

8: $\text{ask est} = (1 - \alpha) * \text{ask est} + \alpha p$

9: $\text{bid est} = (1 - \alpha) * \text{bid est} + \alpha p$

10: ELSE (there is a hi bid–lo ask spread)

11: $\text{ask est} = (1 - \beta) * \text{ask est} + \beta \text{lo ask}$

12: $\text{bid est} = (1 - \beta) * \text{bid est} + \beta \text{hi bid}$

13: ENDIF

The approach RoxyBot uses for estimation of the entertainment ticket auctions is based on setting some variables during the game $\alpha = 0.1$ and $\beta = 0.05$.
Also, two internal price estimates are maintained for all entertainment tickets, an ask_est and a bid_est.
The adjustment of these estimates is made in the direction of the trade price, if any trade takes place. Else, if a bid-spread occurs, the ask_est is adjusted in the direction of lo_ask and bid_est is adjusted in the direction of hi_bid.

3.4 Future directions

As playing the same method became quite intuitive and the market had new requests, another type of game was initiated.
The SCM game tournament started back in 2003, as a simple competition.
From the 2007 edition, two more challenges where added: a Procurement Challenge and a Prediction Challenge.
Different extensions were made from year to year.
Agents are simulations of small manufacturers, who must compete with each other for both supplies and customers, and manage inventories and production facilities[12].

Supply chain management is concerned with planning and coordinating the activities of organizations across the supply chain, from raw material procurement to finished goods delivery. In today's global economy, effective supply chain management is vital to the competitiveness of manufacturing enterprises as it directly impacts their ability to meet changing market demands in a timely and cost effective manner. With annual worldwide

supply chain transactions in the trillions of dollars, the potential impact of performance improvements is tremendous[12].

While today's supply chains are essentially static, relying on long-term relationships among key trading partners, more flexible and dynamic practices offer the prospect of better matches between suppliers and customers as market conditions change. Adoption of such practices has however proven elusive, due to the complexity of many supply chain relationships and the difficulty in effectively supporting more dynamic trading practices[4].

TAC SCM was designed to capture many of the challenges involved in supporting dynamic supply chain practices, while keeping the rules of the game simple enough to entice a large number of competitors to submit entries. The game has been designed jointly by a team of researchers from the e-Supply Chain Management Lab at Carnegie Mellon University and the Swedish Institute of Computer Science (SICS) [4].
Six agents compete in each game. The game takes place over 220 TAC days, each day being 15 seconds long. The agent with the highest sum of money in the bank at the end of the game is declared the winner. The format and content of the various messages exchanged between the agents and the game server are available in the software documentation.
In addition to the interaction with the suppliers and customers, agents (and game viewers) have access to other data within a game[4].

TAC-SCM Procurement Challenge (SCM-PC):
The challenge requires agents to manage supply chain risk by negotiating long-term, quantity flexible procurement contracts and supplementing these contracts with one-off procurement orders. As such, this challenge complements the current "baseline" TAC- SCM scenario by extending the space of procurement options available to supply chain trading agents.
Specifically, manufacturer agents will rely on a combination of:

- Long-term "quantity flexible" contracts. These contracts specify minimum component quantities a manufacturer agent commits to purchasing weekly (at a fixed price) from a given supplier agent and include options to increase these quantities by up to some percentage (at the same fixed price).
- One-off contracts. These are the same supply contracts as the ones negotiated in the baseline TAC-SCM scenario[12].

The TAC-SCM Procurement Challenge (or "SCM-PC") game simulates D days of operation (where D = 100 days). It features n manufacturer agents (where n = 3) competing for supply contracts from 8 different supplier agents every supplier offers both long-term and one-off contracts. Long-term contracts are negotiated at the start of the game and last for the game's full duration. Each week, manufacturer agents may decide to order more than the minimum quantities they committed to up to a pre- specified max quantity. Each day, they may also decide to procure additional components outside of their long-term procurement contracts (specifying quantity and delivery date) [12].

TAC-SCM Prediction Challenge (SCM-PC):
In order to effectively manage a supply chain, a TAC SCM agent must be capable of performing a number of interrelated tasks. In the TAC SCM Prediction Challenge, agents will be evaluated on their ability to perform a single one of these tasks in isolation: making predictions about prices. In particular, agents will make daily predictions over the course of a number of TAC SCM games about four different types of prices: current and future computer prices, and current and future component prices[12].
The predictions consist of:

1. The price at which each RFQ sent from customers on the current day will be ordered (i.e., the lowest price that will be offered by a manufacturer),
2. The median price at which each of the 16 types of computer will sell in Ncomputer days,
3. The price that will be offered for each RFQ sent by the PAgent to a supplier on the current day, and
4. The price that will be offered for each of a number of provided RFQs that will be sent

by the PAgent to suppliers in Ncomponent days[12].

In order to allow games to be followed in real time, and also analyzed in depth at a later date, an additional set of metrics (including the following) will be monitored throughout the game. These metrics are used by the game viewer to provide a visual representation of the game as it proceeds, and are stored within the game logs for post mortem analysis[12].

- Bank balance
- Inventory quantities and cost of inventory held
- Delivery performance
- Assembly cell utilization
- All RFQ, offers and orders exchanged by agents, customers and suppliers[12].

Note that this information is not provided to the agents directly, and agents should not attempt to access it though external means (i.e. through the game viewer or the game logs). The use of such external information, either manually or automatically, is regarded as external 'tuning' of the agent. As such, according to the existing competition rules, it is forbidden within any specific round during the finals of the competition[12].

4 Conclusion

The most interesting phase was seeing all these agents competing on one game. The results are very important, as the three strategies are very well defined, but all have their leaks.

After an 80 rounds game, the following results have been obtained by the three agents. This is presented in Figure 1.

The names are abbreviated in the pictures:

Rox : RoxyBot

Wal : Wlaverine

Mer : Mertarcor

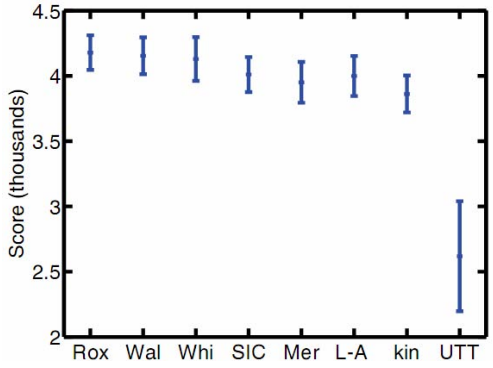

Fig. 1. Graphic with the results obtained by the agents

The place obtained by Mertarcor is not so good.

In order to explain this situation, the steps in the game have to be observed, by category for bidding.

TABLE 2. Game bidding steps

	Rox	Wal	Mer
Hotel Bids	130	81	94
Average Hotel Bids	170	115	147
Won Hotels	15.99	16.79	18.44
Hotel Costs	1102	1065	902
Unused Hotels	2.24	1.82	4.86
Hotel Bonus	613	598	590
Trip Penalty	296	281	380
Flight Costs	4615	4655	4834
Event Profits	110	26	123
Event Bonus	1470	1530	1369

As expected, the best result obtained by Mertarcor was for won hotels. Their strategy is best for hotels. But only for won hotels, as they do not use both hotels all the time, they are oriented only on one hotel. This is why the agent does not obtain a maximum bonus for hotels.

RoxyBot uses both hotels, not as well as Mertarcor, but it gets the maximum bonus.

Walverine has a poor strategy for hotels, as it appears here.

Basically, for all goods, Mertacor has a one-target oriented strategy, while RoxyBot and Walverine have a all-goods oriented strategy. Although Mertacor orients towards the most expensive goods in order to obtain better prices and more money, RoxyBot and Walverine manage to use most of the goods at lower prices, that's why they get also bonuses, and their rank increases.

Finally, RoxyBot has the most equilibrate strategy, not very strong on one good, but sufficiently strong on all goods. So if it bids on one good, it may not get the best price, but at least the immediate one after the best price.

Mertarcor obtains the best price for some goods, while for others it obtains very poor prices, being also penalized because of its orientation towards only some goods.

References:

[1] [1] A. Greenwald and J. Boyan, "Bidding Under Uncertainty" Theory and Experiments. In Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence. (Jul. 2004), 209-216.

[2] [2] M.P. Wellman, D.M. Reeves, K.M. Lochner, Y. Vorobeychik, "Price prediction in

a trading agent competition". Journal of Artificial Intelligence Research 21 (2004) 19–36.

[3] [3] D. Kehagias, P. Toulis, P. Mitkas - "A Long-Term Profit Seeking Strategy for Continuous Double Auctions in a Trading Agent Competition", SETN'06, May 18-20, 2006.

[4] [4] www.sics.se/tac/

[5] [5] P. Toulis, D. Kehagias and P. A. Mitkas- "Mertacor: A Successful Autonomous Trading Agent", AAMAS'06, May 8-12, 2006.

[6] [6] S. J. Lee, A. Greenwald, and V. Naroditskiy -"RoxyBot-06: An (SAA)2 TAC Travel Agent",pp 1378-1383 IJCAI-07.

[7] [7] M. P. Wellman, D. M. Reeves, K. M. Lochner, and R. Suri- "Searching for Walverine 2005", IJCAI-05, August 1, 2005.

[8] [8] S.-F. Cheng, E. Leung, K. M. Lochner, K. O'Malley,

[9] D. M. Reeves, L. J. Schvartzman, and M. P. Wellman- "Walverine: A Walrasian Trading Agent", Decision Support Systems 39 (2005) 169–184.

[10] A. Greenwald, J. Boyan, "Bidding Algorithms for Simultaneous Auctions: A Case Study ",Journal of Autonomous Agents and Multiagent Systems, Springer, 10(1):67-89, 2005

[11] W. Hildenbrand and A. P. Kirman, "Introduction to Equilibrium Analysis: Variations on Themes" by Edgeworth and Walras, North-Holland Publishing Company, Amsterdam, 1976.

[12] The Supply Chain Management Game for the 2007 Trading Agent Competition, December 2006