# Classification Driven Detection of Opportunistic Bids in TAC SCM*

**Anuj Toshniwal**
Center for Data Engineering,
International Institute for
Information Technology (IIIT).
Hyderabad, India.
anuj.t@students.iiit.ac.in

**Kuldeep Porwal**
Center for Data Engineering,
International Institute for
Information Technology (IIIT).
Hyderabad, India.
kuldeep.porwal@students.iiit.ac.in

**Kamal Karlapalem**
Center for Data Engineering,
International Institute for
Information Technology (IIIT).
Hyderabad, India.
kamal@iiit.ac.in

**Abstract**

The main objective of a bidding agent in TAC SCM is to get profitable orders and to get enough orders to keep the production going. There is a delicate balance that the bidding agent needs to maintain while deciding on which specific orders to bid and what bidding price to set. In this highly complex bidding problem with (i) many inter-dependencies, (ii) multiple information flows, (iii) historical data and knowledge, the bidding agent can bid for a few opportunistic orders at a reasonably higher price, which gives higher profit. In this paper, we use classification to determine opportunistic bids to increase our profit. Our solution is robust and adapts according to the dynamic changes in the market condition and the competition provided by the competing agents. Our results show that classification using our opportunistic approach contributes to a significant percentage of our agent's profit.

## 1 Introduction

In today's highly competitive market place, managing supply chains is one of the most challenging problems. Supply chains consist of heterogeneous subsystems and complex relationships requiring collective effort and constraint based optimization. In dynamic market conditions, it is very difficult to take both short and long term decisions simultaneously. A multi agent system, consisting of several autonomous agents can address this problem and take appropriate decisions.

The Trading Agent Competition for Supply Chain Management (TAC SCM) (Collins et al. 2007) was designed to capture many of the challenges involved in sustaining dynamic supply chain practices. This paper describes the bidding strategy of the *Iota* trading agent, which is one of the *winning* agents in TAC SCM 2012. In this paper we focus on the bidding sub-system, which is a crucial component. Bidding in TAC SCM can be considered similar to First Price Sealed Bid Auction in which the submitted bids against customers' request for quotes (RFQs) are compared and the bidder with the lowest bid wins the order.

*Trading Agent Competition for Supply Chain Management


Agents participating in the game must simultaneously compete in two markets with interdependencies, and take their decisions based on incomplete information about the state of the market. In one market, agents buy their supplies and in the other, they sell their finished products. Supply and demand of each market varies dynamically not only by randomness but also due to competing agents' strategies. Agents have a limited time to take a number of decisions. All these factors make the problem quite challenging to address.

In this highly complex bidding problem with (i) many interdependencies, (ii) multiple information flows, (iii) historical offline data and knowledge, the *agent can bid for a few opportunistic orders at a reasonably higher price, which gives higher profit.*

We have modeled our agent as a *business entity* capable of using game theory, machine learning and data mining techniques. Our agent focuses on dynamics such as market trend, customer demand and changes in procurement costs to learn and adapt. We present a classification driven approach to identify opportunistic bids. Our agent classifies the customer requests received each day into different classes using a parameterized decision tree and bids accordingly.

The remaining paper is structured as follows. In section 2, we discuss TAC SCM game specifications and describe the problem and related work. We present an overview of our agent's bidding module and focus on classification of customer requests in section 3. In section 4, we present the results showing performance of our agent in TAC SCM.

## 2 Bidding in TAC SCM

In this section, we describe the game overview and problem specifications. We also highlight some relevant related work that has been done on the problem.

### TAC SCM Overview

In TAC SCM, six assembly agents compete in a simulated market environment constituting a supply chain. An agent's task is to procure components, manufacture personal computers (PCs) and to win customer orders (bidding subtask).

Each simulation day is of 15 sec and the game consists of $E$ days (in standard settings, $E$=220). At the start of each day, all agents receive a Request For Quotes (RFQ) bundle specifying customer requests. Each RFQ specifies (i) prod-

uct id, (ii) desired quantity, (iii) due date, (iv) reserve price and (v) penalty.

An agent must bid to satisfy both quantity and due date for the customers to consider the bid. Also, each day the agents receive a price report containing information about the lowest and highest prices for each type of PC ordered on the previous day. Full specification of the TAC SCM game is given in (Collins et al. 2007).

## Problem Specification

The agent receives numerous RFQs from the customer each day, requesting different types of PCs. The total quantity of PCs requested on a given day is much higher compared to an individual agent's production capacity. Faced with this problem of plenty, the agent aims to identify and bid on a set of selected customer RFQs that maximize the profit subject to its factory capacity constraints (Collins et al. 2007) and component (current and expected in future) availability.

The profit depends not only on the RFQs being bid on the current day, but also on RFQs that need to be bid in immediate future. If these future RFQs are ignored while selecting the current day's bids, the agent might commit all available production resources on the current RFQs, leaving it unable to bid on more profitable future RFQs.

In a naive approach, the agent attempts to predict the highest winning price and bid on selected RFQs, expecting to win each one of them. But we follow a more viable opportunistic approach, which involves placing high bids on a subset of RFQs depending upon current market situation, expecting to win a few of them. Our agent tries and ensures that it balances this risky approach in such a way that it does not lead to low revenue while keeping its assembly line busy.

**Design Challenges** There is a delicate balance that the agent needs to maintain while deciding on which specific orders to bid, and what bidding price to set. The agent does not want to acquire all orders but neither does it wish to underutilize its factory production capacity. Agents in the past have used varying approaches to solve this problem, but these approaches are highly complex and involve expensive computations.

## Related Work

Different techniques have been explored by agents to solve Bidding problem. Mostly statistical approaches are used to win sealed price based auctions. Some of the those approaches are discussed in the following survery (Vicky and Cassaigne 2000). A heuristic approach to search for the optimal set of bids is described in (Pardoe and Stone 2004). An approximate optimization procedure that uses values to decompose the decision problem is presented in (Kiekintveld et al. 2009). Other examples of some important work include empirical game theory approach (Jordan, Kiekintveld, and Wellman 2007), mathematical programming approach [(Benisch et al. 2004) and (Kovalchuk and Fasli 2008)], decision making [(He et al. 2006), (Collins, Ketter, and Gini 2009), (Ketter et al. 2009)], particle swarm optimization and policy search via adaptive functions (Chatzidimitriou, Symeonidis, and Mitkas 2012). Main disadvantage for most of these approaches that are based on learning from offline past bid data is that there are many dependent variables which make it difficult for the agent to map the current situation to a similar situation in the past data. There are numerous variables like competing agents, past and present market trend, agent performance, expected future demand, supplier factory settings - price, delivery time, capacity - and resource availability (component and factory cycles), which affect the decision making of the agent. It is very difficult to map all these parameters and settings to some previous game data and use it. Further, most of the models also fail to include current rivalry in the market while some algorithms fail to adapt quickly. In this paper we present a classification model which captures dependencies and can easily adapt to any runtime changes.

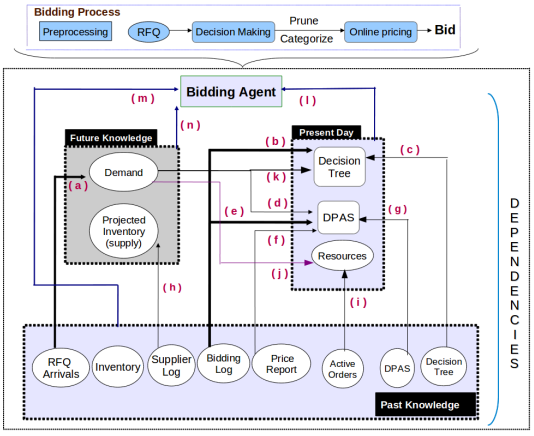


Figure 1: Dependency Model for Bidding subsystem.

# 3 Bidding Agent

We have modeled our agent as a *business entity* capable of using game theory, machine learning and data mining techniques. Our agent focuses on dynamics such as market trend, customer demand and changes in procurement costs to learn and adapt. In this paper, we present only the classification subtask of the bidding subsystem.

Each day, the agent takes decisions based on past, current and expected future dynamics of the market. Our agent considers the daily market condition as a data stream and uses the information from the past $k$ days window (we use $k$=5). We follow trial and error approach with different window sizes to figure out the corresponding impact on agent's performance and fix the window size accordingly. Window size is an important parameter in our strategy, since our agent learns the current market situation based on information gathered in present window and adapts accordingly. Window size *not only* affects the bidding module, but also plays a crucial role in the pricing and procurement modules (which are *not* discussed in this paper).

In Figure 1, we describe the dependencies between various subtasks of bidding. These dependencies are described below:

Table 1: Daily Execution Loop for bidding

(i) At the start of each day, the agent does preprocessing of data involving:
- Demand Estimation.
- Price Report Analysis.

(ii) Classify the RFQs. Prune the non-profitable RFQs and *sort* the remaining RFQs.
(iii) Process the *sorted* RFQs, predict a bid price for each RFQ and send offer.

(a): Based upon the current arrivals of RFQs, agent predicts the behavior of future RFQ arrivals and demand (section 3).
(b), (c), (k): Our agent's decision making criterion depends upon its previous criterion, it's bidding performance and future demand (section 3).
(d), (e), (f), (g): Dynamic Profit Adjuster Scale (DPAS) is our price prediction module, used to decide the bid price. Our agent's pricing strategy depends upon it's previous state, bidding log, analysis of price reports and expected future demand.
(i), (j): Current availability of resources depends not only on current active orders, but also on demand in future.
(h): Projected components inventory of agent depends upon the past negotiations with suppliers.
(l), (m), (n): The bidding agent learns from the past, present and future knowledge as shown in Figure 1.

Table 1 describes the daily execution loop for bidding followed by our agent. Detailed description of pricing strategy and supplier negotiation strategies are not described in this paper.

## Demand Estimation

Demand estimation is necessary to make important decisions like classification of RFQs, planning for future inventory levels and resource restriction.

Products are classified into three market segments – high range, mid range, and low range – according to their nominal price (Collins et al. 2007). Correspondingly, customers exhibit their demand by issuing RFQs in each segment. Number of customer RFQs for day *d* issued in each market segment is determined by a draw from a Poisson distribution with mean $Q_d$. Demand evolves according to $Q_d$, which is varied using a trend $\tau$ given by daily update rule:

$$Q_{d+1} = min(Q_{max}, max(Q_{min}, \tau_d Q_d)) \quad (1)$$

$\tau$ is updated according to a random walk given by:

$$\tau_{d+1} = max(\tau_{min}, min(\tau_{max}, \tau_d + random(-0.001, 0.001))) \quad (2)$$

Whenever $\tau$ exceeds its min-max bound, it is reset to its initial value, which is equal to 1. The states $Q$ and $\tau$ are hidden, but we know that $Q_{d+1}$ is a deterministic function of $Q_d$ and $\tau_d$ as specified in Equation 1. This deterministic relation is exploited using a Bayesian Network Model in (Kiekintveld et al. 2004) to estimate the demand process based on the current observed demand. We use this distribution to estimate expected future demand as in (Kiekintveld et al. 2004).

We require a holistic view of demand. For this, we consider the sum of RFQs of the three market segments ($Q_{sum}$) and define net demand of a day *d* ($D_{net}(d)$) as follows:

$$D_{net}(d) = \begin{cases} low, & \text{if } Q_{sum}(d) < 130. \\ mid, & \text{else if } Q_{sum}(d) < 220\,. \\ high, & \text{otherwise.} \end{cases} \quad (3)$$

The above values used for comparison are chosen through empirical analysis of large number of games and by analyzing the number of RFQs sent by customers each day which falls in the range of 80 to 320 RFQs per day.

## Price Report Analysis

We define $H_{(d,p)}$ and $L_{(d,p)}$ for a product *p* at day *d* from High Price ($HighPrice_{(d,p)}$) and Low Price ($LowPrice_{(d,p)}$), received in price report at the start of each day. We use $H_{(d,p)}$ and $L_{(d,p)}$ to estimate the current trend and predict bid price accordingly.

Since we do not want to bias our decision on price fluctuation of a single day, we define $H_{(d,p)}$ as the weighted sum of High Prices of past *k* days window. Exponentially decaying weight factor ($\gamma/2^i$) is used to give higher weightage to recent days.

$$H_{(d,p)} = \sum_{i=1}^{k} \frac{\gamma}{2^i} * HighPrice_{(d-i,p)} \quad (4)$$

Here, $\gamma$ is the scaling factor such that ($\gamma * \sum_{i=1}^{k} 2^{-i} = 1$). Similarly we find $L_{(d,p)}$ based on Low Prices of past *k* days.

If we increase the window size, the weight associated with most recent information decreases as the weight is distributed to include more days. This leads to slow adaptation by our agent which in turn affects our agents performance. Similarly decreasing the window size will bias our agent's decision towards most recent information leading to poorer performance, as shown in Results (section 4) under Experiments subsection.

## RFQ Classification

The vital aspect for winning any multiplayer game is that the agent must be alert to identify and grab any opportunity that comes along. Sometimes taking immediate rewards are better than waiting for long term rewards. On the other hand, sometimes the agent must think long term and be willing to take a dip in performance if it leads to a spike later, thereby improving the agent's performance. An opportunistic agent is one which is able to identify opportunities and act positively on them. But sometimes there are cases when there is no opportunity. Then agent should be flexible enough to choose the best possible strategy for that period too.

To identify and utilize opportunities, we classify the RFQs and bid accordingly,i.e. the agent predicts different bid price based on the assigned class of the RFQ to maximize it's profit. We use a parameterized decision tree model along with other learning algorithms like dynamic programming and heuristic based learning to solve this problem as elaborated below. We dynamically learn the parameters of the

decision tree based on current agent performance and market conditions of past, present and future.

With the use of decision tree we prune the non-profitable RFQs and classify the remaining RFQs into three different classes namely *Singles*, *Fours* and *Sixes* [1] .

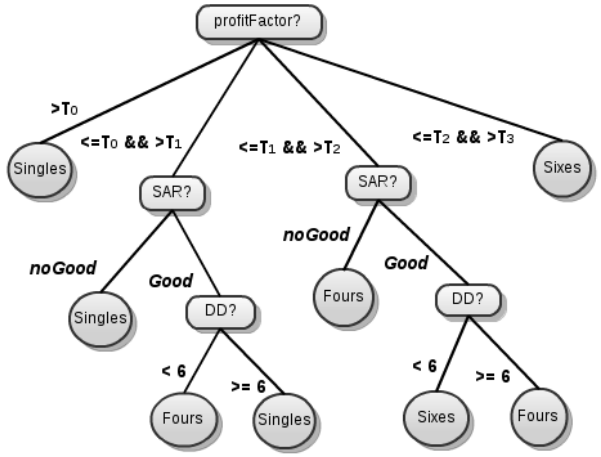


Figure 2: Decision Tree for RFQ Classification

• **Singles**: RFQs which are highly profitable, considering the manufacturing cost incurred by the agent. We need to try and acquire orders for these RFQs, even if we need to bid at a slightly lower price than desired, just to be sure that we win the bid. The profit margin, even at this lower price, is better than most other RFQs. The intuition here is that other agents might also try to bid on these RFQs and we must aim to outbid them.

• **Fours**: These are the RFQs for which our agent takes some risk. The agent bids on these RFQs, expecting to win a few of these. Such RFQs corresponds to mainly two cases – (i) the expected profit margin compared to other RFQs is low, and (ii) profit margin is decent but the future demand is expected to increase, implying that the agent can earn more in the future. In either case, the agent takes risk knowingly and bids at a comparatively higher price in order to earn better profit. Also based on the acceptance rate of these bids, our agent analyzes the current pricing trend and competition in the market.

• **Sixes**: Remaining RFQs are classified as *Sixes*. We try for maximum profit and bid very close to the reserve price for such RFQs. Orders for such RFQs are like bonus -it is good if we have them, but it does not matter if we don't.

As we move from *Singles* to *Fours* to *Sixes*, the element of risk taken by the agent increases. In other words the chances of agent winning the order decreases. A 'steady flow' of orders corresponding to *Singles* RFQs is a must to ensure decent profit and sufficient factory utilization. Classifying all RFQs into Singles or Fours or Sixes is considered to be the pure strategy as each class has its own significance. But in SCM, market conditions are very dynamic and to adapt and utilize these conditions we have to use the proper mix of Singles, Fours and Sixes i.e. we have to be opportunistic.

We did thorough analysis to determine certain rules for building decision tree and for this we ran 1261 games with various parameters. These rules help us in selecting the attributes and in deciding the splitting conditions for various attributes of the decision tree.

**Empirical Analysis based Classification** After empirical analysis of 1261 games, we came up with certain heuristics and threshold values of the various parameters about the game. Using this we built the decision tree model. The decision tree is parameterized, where each parameter is learned / adapted dynamically based on market conditions of past, present and future. Thus, this decision tree is applicable for all types of games low, mid and high demand. Described below are the basic rules that we have obtained through analysis.

*Rule 1* : We increase the number of *Fours* and *Sixes* when either there is less competition in the market i.e. *Singles* acceptance rate is *Good*, or if the future demand is expected to be high. In the former case, the agent has the opportunity to take more risks; in the latter the agent knows it will get the opportunity to sell it's products at a comparatively higher price in the future, thereby bidding for more *Fours* and *Sixes*.

*Rule 2* : We decrease the number of *Fours* and *Sixes* when either there is high competition in the market i.e. if *Singles* acceptance rate drops below a certain level (*noGood* condition), or if the future demand is expected to be low. In both cases, we need to try and get more orders to increase revenue.

*Rule 3* : Closer the due date, the more difficult it is for the agents to fulfill the order in time, leading to possible late deliveries and penalty. Hence, RFQs with closer due date are classified as *Fours* or *Sixes* and priced a bit higher than normal. But if the expected profit is very good and resources are available, we take the risk with respect to late delivery and classify the RFQ as *Singles*.

**Decision Tree** Decision tree (Figure 2) has 3-attributes: $<$ *ProfitFactor*, *DD*, *SAR*$>$.

(i) *ProfitFactor* is used to consider the profitability feature of RFQs. It is the main attribute in classification. To define *ProfitFactor*, we first define *PriceFactor* for a RFQ as follows:

$$PriceFactor = \begin{cases} H_{(d,p)} - CostPrice_p, & \text{if Res} > H_{(d,p)} \\ Reserve - CostPrice_p, & \text{if Res} < H_{(d,p)} \end{cases} \quad (5)$$

Here $d$ is current date, $p$ is productID and *Res* is the reserve price of the RFQ. $CostPrice_p$ represents the minimum cost price of the given product, determined from the projected component inventory of agent available before the due date of the RFQ. We first find those set of components

[1]We use *Cricket* game analogy – *Singles* are used to keep the scoreboard ticking, while *Fours* and *Sixes* are highly risky but opportunistic.

available in our projected inventory which sums up to minimum nominal price for the product and this price is considered as $CostPrice$. Say, if the due date for the RFQ is day 56. We consider components required for assembling the product, which are expected to be available by day 54 (one day is for production and the last day for delivery). From this set we find the components available with minimum cost and calculate the *CostPrice* of product by adding the price of all 4 corresponding components.

We use *minimum* cost price to determine the *best* possible profit that we can get for each RFQ, by considering the inventory available. To normalize the *PriceFactor* corresponding to each RFQ we define *ProfitFactor* as follows:

$$ProfitFactor = PriceFactor/AssemblyCyc_p \quad (6)$$

$AssemblyCyc_p$ is the number of assembly cycles required to manufacture a unit quantity of the product.

(ii) *DD* attribute considers the due date of the RFQ. Due date plays an important role in pricing and resource allocation.

$$DD = (dueDate - currentDate) \quad (7)$$

(iii) *SAR* represents offer acceptance rate of *Singles* RFQs over previous *k* days. This attribute is used to evaluate the current market performance of the agent and the rivalry between the competing agents. High value of *SAR* implies that the agent is performing well by efficiently handling the competition from the rival agents and vice versa. We consider RFQs of only *Singles* category and not *Fours* or *Sixes* category for this factor, because of the critical importance of acquiring orders corresponding to *Singles* RFQs.

$$SAR = (singlesOrdered/singlesOffered) \quad (8)$$

where $singlesOffered$ is the sum of offers sent for RFQs treating them as *Singles* in previous *k* days and $singlesOrdered$ is the sum of orders received corresponding to these offers.

To implement the discussed rules in the decision tree we have to dynamically learn the thresholds to split *profitFactor* attribute.

**Determining Thresholds** RFQs with *ProfitFactor* value less than $T_3$ are pruned. $T_0$, $T_1$, $T_2$ and $T_3$ thresholds are dynamically learned based on the following two factors:
(i) expected customer demand in the future (*futureD*). It is computed as follows:

$$futureD = mode(D_{net}(i)) \quad i = (d+1)\ to\ (d+k) \quad (9)$$

If the predicted future demand is higher than current observed demand, we increase the thresholds, since price is expected to increase when the demand improves and vice versa. Thus, if higher demand is expected in future, we can wait for the prices to rise, and at this time take more risks by bidding for *Fours* and *Sixes* RFQs. If lower demand is predicted in future, agent must try and acquire more orders now, before prices start decreasing. We define *demandEffect* for current day *d* as:

$$demandEffect = \begin{cases} -1, & \text{if } futureD < D_{net}(d). \\ 1, & \text{if } futureD > D_{net}(d). \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

(ii) offer acceptance rate of *Singles* RFQs (*SAR*). If *SAR* drops below a predefined lower bound static threshold *lowT*, we need to decrease the value of thresholds to try and get more orders. Similarly, if *SAR* is greater than upper bound threshold *highT*, we can afford to increase the thresholds and convert few RFQs from *Singles* to *Fours* and some RFQs from *Fours* to *Sixes* in order to get more profit. We have set *lowT* and *highT* values on the basis of a large number of training games' result logs. We define *sarEffect* as:

$$sarEffect = \begin{cases} -1, & \text{if SAR<lowT.} \\ 1, & \text{if SAR>highT.} \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

For day *d*, we compute $T_i$ by using the following dynamic programming approach:

$$T_i(d) = (1 + (\beta * (sarEffect + demandEffect))) * T_i(d-1) \quad (12)$$

Here, $\beta$ is the rate of change of threshold. $\beta$ is a constant factor in the range [0,1]. Value of $\beta$ is decided based on the confidence in determining *demandEffect* and *sarEffect* and is given by

$$\beta = 0.5 * \frac{TotalSingles}{TotalRFQs} + 0.5 * Pr(futureD|D_{net}(d)) \quad (13)$$

Here the *TotalSingles* denotes total number of *Singles* offered by agent and *TotalRFQs* denotes total number of RFQs received by agent in the current window. The probability term is determined through Demand Estimation process described above.

Once we update $T_i$*'s*, we wait for a period of 3 days before making another update, to give time for the change in $T_i$*'s* to work.

After classification, we first process the *Singles*, then *Fours* and lastly *Sixes*. In each category, the RFQs are ranked in descending order based on *ProfitFactor* of the RFQs. This method of ranking ensures that the profitable RFQs come out on top and are given more priority when allocating resources.

The agent has already committed some factory cycles in the future for the completion of it's active orders. Further, depending on the future demand, some of the factory cycles are restricted and not available for committing on the current day. For each RFQ, we check if sufficient free factory cycles are available to complete the order. If not, we check for any extra inventory which can be committed to the particular RFQ. But, if still unsuccessful, we demote the category of RFQ – from *Singles* to *Fours*; *Fours* to *Sixes*; *Sixes* to Prune.

Also to accomplish *Rule 1* and *Rule 2*, we convert some RFQs from *Fours* to *Singles* and some from *Sixes* to *Fours*. The intuition here is that some profit is better than no profit. But, we also ensure that we maintain the balance, as ultimately it is the profit that matters at the end of the game and not the revenue.

Our agent is flexible enough to know when there isn't any opportunity available. In that case, we can intuitively infer that rivalry is strong and it is better to increase number of *Singles*.

| | Strategy | Profit% | Utilization (in %) | *Singles* | | | | *Fours* | | | | *Sixes* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | True | False | Offers | Orders | True | False | Offers | Orders | Offers | Orders |
| 1. | Only *Sixes* | 12.29 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 31681 | 429 |
| 2. | Only *Fours* | 24.71 | 34 | 0 | 0 | 0 | 0 | 7154 | 20092 | 27246 | 2008 | 0 | 0 |
| 3. | Only *Singles* | 58.38 | **71** | 15030 | 5887 | 20917 | **6524** | 0 | 0 | 0 | 0 | 0 | 0 |
| 4. | Window (=1) | 58.79 | 57 | 15495 | **1290** | 16785 | 4474 | 5975 | 296 | **6271** | **734** | 2839 | 205 |
| 5. | Window (=3) | 62.01 | 58 | 15909 | 1062 | 16971 | 4592 | 6118 | 208 | **6402** | 878 | 3639 | 308 |
| 6. | Window (=7) | 61.64 | 60 | 16062 | **1152** | 17214 | **4501** | 6389 | **224** | 6629 | 833 | 3198 | 292 |
| 7. | Window (=9) | 57.09 | 58 | 17980 | **1323** | 19303 | 4311 | 6634 | **308** | 6942 | 796 | 3402 | 187 |
| 8. | Opportunistic Bidding | **66.68** | 64 | 16690 | 845 | 17535 | **4812** | 4751 | 152 | **4903** | **1170** | 3651 | **443** |

Table 3: Experiment Results (average of 50 games each)

| Rank | Agent | Average Score (final) | Average Score (at end of $195^{th}$ day) |
|---|---|---|---|
| 1. | Deep Maize | 11.46 M | 8.118 M |
| 2. | Punga2Bani | 8.364 M | 2.635 M |
| 3. | Iota | 5.835 M | 3.226 M |
| 4. | HEU2012 | 4.223 M | -4.058 M |
| 5. | Mertacor | -1.367 M | -5.586 M |
| 6. | Crocodile | -2.625 M | -7.897 M |

Table 2: Average scores of Final round in TAC SCM 2012

# 4 Results

In this section, we evaluate the agent performance and compare the agent strategies under different experimental setups.

## Agent Performance

To evaluate and validate the adaptive and dynamic techniques of our agent under different conditions, we took part in TAC SCM 2012 competition. The final score provides suggestive evidence regarding the efficacy of our strategy and approach. Our agent *Iota* finished third behind two other agents. This is mainly because our agent was not able to adapt well in the end phase of the game and fell behind the other top agents after 195th day. Table 5 shows that at the end of 195th day, the average profit of our agent was second highest. But during the end phase of the game, we were unable to hold our position and slip to third. To improve our performance in the end phase of the game, we need to improve coordination between the different modules of our agent. Apart from bidding module, pricing and procurement modules also play an important role.

In this paper, we focus only on the classification sub-task of the bidding subsystem. Pricing and procurement strategies are not discussed.

## Experiments

We present the results of controlled experiments designed to compare different strategies of our agent and their impact on our agent's performance. We run 50 games for each experiment *against five of the best agents that have participated in TAC SCM competition* – TacTex (Pardoe and Stone 2004), DeepMaize [(Kiekintveld et al. 2009), (Kiekintveld et al. 2004)], Phant Agent (Stan, Stan, and Florea 2006), Maxon and Mertacor (Chatzidimitriou, Symeonidis, and Mitkas 2012). All five agents have finished first in atleast one of the previously held TAC SCM competitions. We use the binary codes of the agents available in TAC Agent Repository to run the simulations. Running each experiments 50 times basically covers different low demand, mid demand and high demand games and gives us a nice platform to analyse these results.

For the experiments conducted, Table 3 presents the comparison of average results of different parameters of our agent. Comparison is done mainly on the basis of *Profit%* among other factors.

$$Profit\% = \frac{Profit_R}{\underset{\forall i \in agents}{max}\ (Profit_i)} * 100 \qquad (14)$$

$Profit_R$ indicates the profit of our agent and $Profit_i$ denotes the profit of agent *i*.

In Table 3, *False* indicates the number of RFQs which should have been classified differently and bid at a different price. If majority of the agents bid for a RFQ at a price, which is either 10% greater or 10% lesser than our agent's bid price, we term the RFQ as falsely classified. Lesser number of *False* offers indicates better strategy. Complement of the *False* set is the *True* set. *True* offers are the offers corresponding to the RFQs which are classified and bid appropriately.

In the following experiments, we test the impact of our RFQ classification model on our agent performance. Strategy 8 is the final version of our agent, which we entered in the competition. We use the mixed strategy for Singles, Fours and Sixes RFQs classified using our parameterized decision tree. Window size is 5.

In strategy 1, we treat all the RFQs as *Sixes* and bid. Similarly in strategy 2 and strategy 3, we treat all the RFQs as *Fours* and *Singles* respectively and bid. Strategy 1,2 and 3 are the pure strategies. Strategy 8 is our opportunistic bidding agent that does proper classification and is the mixed Strategy of Strategies 1,2 and 3.

In 'Only *Sixes*' strategy (highly risky) and 'Only *Fours*' strategy (risky), profit and utilization are both less understandable. While in 'Only *Singles*' strategy (safe), the factory utilization is bit high, but profit is less compared to that in strategy 8, representing classification based on identifying opportunity. This is because the agent can maximize its profit by bidding reasonably higher on some RFQs, depending on the situation, expecting to win a few such orders.

In strategy 4, 5, 6 and 7, we change the 'window' size for

| Rank | Agent | Average Score |
|---|---|---|
| 1. | TacTex | 8.49 M |
| 2. | DeepMaize | 6.62 M |
| 3. | Iota | 5.84 M |
| 4. | Phant | 5.67 M |
| 5. | Maxon | 3.31 M |
| 6. | Mertacor | 2.15 M |

Table 4: Comparison of our strategy with top agents

| Setup | Profit% | *Singles* | | *Fours* | | *Sixes* | | Utilization (in %) |
|---|---|---|---|---|---|---|---|---|
| | | Offers | Orders | Offers | Orders | Offers | Orders | |
| F(5),S(0) | 66.68 | 17535 | 4812 | 4903 | 1170 | 3651 | 443 | 64 |
| F(4),S(1) | 70.11 | 15934 | 4902 | 5225 | 1286 | 3817 | 522 | 66 |
| F(3),S(2) | 75.36 | 15401 | 5136 | 5452 | 1367 | 4015 | 549 | 69 |
| F(2),S(3) | 83.07 | 15022 | 5201 | 5610 | 1508 | 4341 | 583 | 73 |
| F(1),S(4) | 90.88 | 14680 | 5315 | 5785 | 1587 | 4498 | 611 | 76 |
| F(0),S(5) | 98.79 | 14140 | 5699 | 6069 | 1634 | 4751 | 647 | 80 |
| Dummy(5) | 100 | 13633 | 5908 | 6518 | 1748 | 5051 | 706 | 93 |

Table 5: Illustrating effect of rivalry due to competing agents.

our agent, keeping everything else the same as our opportunistic agent in strategy 8 where window size is set as 5. The results indicate that the agent is more robust with 5 as the 'window' size. Keeping the window size small makes our agent biased towards a small change in market conditions. On the other hand, the agent adapts slowly to the dynamically changing market conditions if window size is large.

Table 4 illustrates the performance of our *Iota* agent against the five best agents of TAC SCM. In this experiment we compete against those agents for 50 games with equal distributin of low,mid and high demand games.

## Adaptation

Table 5 shows that our agent learns and adapts according to the changes in market conditions and rivalry. *F(x)* in Table 5 represents *x* different agents, which have been finalists in previous years in TAC SCM competition (same agents as those in Table 4). Similarly, *S(y)* represents *y* different agents, which got eliminated in the initial rounds in previous years SCM competitions while competing against agents of Set-F (*F(x)*) , the agents are Crocodile, Botticelli, kshitij, Grunn and GoBlueOval (also taken from TAC Agent Repository). *Dummy* represents the default agent of TAC SCM, based on random bidding strategy. In short *F(x)* represents *x* self learning agents who provides tougher competition in the market while *S(y)* represent *y* self learning agents who had some startegy but can not provide good competition compare to agents in *F(x)*. So as we increase *x* in *F(x)* we increase the competition in the market and as we increase *y* in *S(y)* we decrease the competition in the market.

As the competition decreases, the opportunity increases and our agent is able to procure more orders. Also percentage of orders corresponding to *Fours* and *Sixes* increases, compared to high competition games. This leads to increase in profit. For each value of *x* and *y* we ran 50 simulations with equal distribution of low, mid and high demand games. Hence from Table 5, we can conclude that our agent adapts according to the competition provided by the fellow competing agents.

## Summary

Table 2, 3, 4 and 5 shows that our solution is able to perform effective opportunistic bidding by not only getting higher profit but also maintaining high utilization. Our agent is also able to adapt dynamic market changes. Thus there is scope for threshold based classification driven solution to address this important problem. The thresholds are computed at runtime to determine the RFQs to be bid. Our opportunistic agent has been able to get significant number of *Fours* and *Sixes* even with the top agents of TAC SCM.

Our classification model can be used for multiplayer game problems involving continuous iterations in a dynamic environment, with feedback from previous iterations. Real Time Scheduling System for Taxi (Glaschenko et al. 2009) is one of the example where we can apply our model.[2]

Future work involves further improving the performance of the agent in the end phase of the game [3]. Also, we must predict the set of possible future RFQs and bid appropriately.

## References


Benisch, M.; Greenwald, A.; Grypari, I.; Lederman, R.; Naroditskiy, V.; and Tschantz, M. 2004. Botticelli: A supply chain management agent. In *ACM Trans. on Comp. Logic*, 1174–1181.

Chatzidimitriou, K. C.; Symeonidis, A. L.; and Mitkas, P. A. 2012. Policy search through adaptive function approximation for bidding in tac scm. In *TADA and AMEC*.

Collins, J.; Arunachalam, R.; Sadeh, N.; Eriksson, J.; Finne, N.; and Janson, S. 2007. The Supply Chain Management Game for the 2007 TAC. In *Technical report CMU-ISRI-07-100,Carnegie Mellon University*.

Collins, J.; Ketter, W.; and Gini, M. 2009. Flexible decision control in an autonomous trading agent. *Electron. Commer. Rec. Appl.* 8(2):91–105.

Glaschenko, A.; ; Ivaschenko, A.; ; Rzevski, G.; ; and Skobelev, P. 2009. Multi-agent real time scheduling system for taxi companies.

He, M.; Rogers, A.; Luo, X.; and Jennings, N. R. 2006. Designing a successful trading agent for supply chain management. In *AAMAS '06*, 1159–1166. New York, NY, USA: ACM.

Jordan, P. R.; Kiekintveld, C.; and Wellman, M. P. 2007. Empirical game-theoretic analysis of the TAC Supply Chain game. In *AAMAS'07*, 193:1–193:8. New York, NY, USA: ACM.

Ketter, W.; Collins, J.; Gini, M.; Gupta, A.; and Schrater, P. 2009. Detecting and forecasting economic regimes in


[2]Demonstration for Taxi scheduling is out of the scope of this paper.

[3]Need to improve coordination between the procurement module and the bidding module of our agent.

multi-agent automated exchanges. *Decision Support Systems* 47(4):307 – 318.

Kiekintveld, C.; Wellman, M. P.; Singh, S.; Estelle, J.; Vorobeychik, Y.; Soni, V.; and Rudary, M. 2004. Distributed feedback control for decision making on supply chains. In *14th ICAPS*, 384–392.

Kiekintveld, C.; Miller, J.; Jordan, P. R.; Callender, L. F.; and Wellman, M. P. 2009. Forecasting market prices in a supply chain game. *Electron. Commer. Rec. Appl.* 8(2):63–77.

Kovalchuk, Y., and Fasli, M. 2008. Adaptive strategies for predicting bidding prices in supply chain management. In *10th ICEC*, 6:1–6:10. New York, NY, USA: ACM.

Pardoe, D., and Stone, P. 2004. Bidding for Customer Orders in TAC SCM. In *In AAMAS'04 Workshop on AMEC VI: Theories for and Enginnering of Distributed Mechanisms and Systems*, 143–157. Springer Verlag.

Stan, M.; Stan, B.; and Florea, A. M. 2006. A dynamic strategy agent for supply chain management. In *SYNASC'06*, 227–232. Washington, DC, USA: IEEE Computer Society.

Vicky, P., and Cassaigne, N. 2000. A critical analysis of bid pricing models and support tool. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 3, 2098 –2103 vol.3.