Policy Search through Adaptive Function Approximation for Bidding in TAC SCM

Kyriakos C. Chatzidimitriou[1,2],
Andreas L. Symeonidis[1,2], and Pericles A. Mitkas[1,2]

[1] Department of Electrical and Computer Engineering,
Aristotle University of Thessaloniki, Greece
[2] Informatics and Telematics Institute,
Centre for Research and Technology Hellas, Greece
mertacor@olympus.ee.auth.gr

Abstract. Agent autonomy is strongly related to learning and adaptation. Machine learning models generated through the use of historical data or current environmental signals, provide agents with the necessary decision-making and generalization capabilities in competitive, dynamic, partially observable and stochastic environments. In this work, we discuss learning and adaptation in the context of the TAC SCM game. We apply a variety of machine learning and computational intelligence methods for generating the most efficient sales component of the agent, dealing with customer orders and production throughput. Along with utility maximization and bid acceptance probability estimation methods, we evaluate regression trees, particle swarm optimization, heuristic control and policy search via adaptive function approximation in order to build an efficient, near-real time, bidding mechanism. Results indicate that a suitable reinforcement learning setup coupled with the power of adaptive function approximation techniques is a good candidate for enabling high performance strategies.

Keywords: adaptive function approximation, trading agent competition, supply chain management, echo state networks, neuroevolution of augmented reservoirs.

1 Introduction

An agent in the trading agent competition (TAC) supply chain management (SCM) game should follow the rule: *"An agent should sell as high as possible and buy as low as possible, while maintaining the highest possible throughput in both factory and inventory (and by throughput we mean to have a high factory utilization, but also sell the produced personal computers and not just store them) and not default on deliveries"* [1]. Based on this context, we focus on the selling and throughput aspects of the agent, in order to improve performance. We thoroughly investigate the daily task of selecting the requests-for-quote to respond to and the prices to offer, given the limited amount of time (near real-time) and bounded resources, and try to provide an optimal decision making solution.

E. David et al. (Eds.): AMEC/TADA 2012, LNBIP 136, pp. 16–29, 2013.

Additionally, within the context of our work we perform adaptive function approximation methodology through neuroevolution of augmented reservoirs (NEAR), in order to adapt the architecture and the parameters of the function approximator to the problem at hand, with little or no human input [2]. We form the selling problem as a standard reinforcement learning [3] control problem and let NEAR find an appropriate policy in the form of an echo-state network (ESN). For evaluation purposes, we benchmark RL approach against machine learning, particle swarm optimization and heuristic control techniques.

The paper is structured as follows: Section 2 provides a rough introduction into TAC SCM game and NEAR. Section 3 discusses the different methods used to approach the problem along with related work for each method. Section 4 presents the experimental setup and the results obtained, along with a brief discussion. Finally, Section 5 summarizes findings and concludes the paper.

2 Background

2.1 TAC SCM

Within the scenario of the TAC SCM game [4,5], each agent participating represents a personal computer manufacturer with limited production capacity. Six such agents compete in selling 16 different types of PCs to potential customers. The agents' tasks are to negotiate on supply contracts, bid for customer orders, manage daily assembly activities and ship completed orders to customers. Every day, customers send Request-For-Quote (RFQ) messages and agents bid on them, depending on their ability to satisfy delivery dates and prices. Bid should not exceed the reserve price the customer requires. The next day, the customer orders from the agent that made the winning offer. The agent must deliver the products on time, otherwise it is charged with a penalty. Winner is declared the agent with the greater bank balance at the end of the game. Game length is 220 simulated days, with each day lasting 15 seconds.

2.2 Echo State Networks

The idea behind *reservoir computing* (RC) and in particular ESNs [6] is that a random *recurrent neural network* (RNN), created under certain algebraic constraints, could be driven by an input signal to create a rich set of dynamics in its reservoir of neurons, forming non-linear response signals. These signals, along with the input signals, could be combined to form the so-called *read-out function*, $y = \mathbf{w}^T \cdot \phi(\mathbf{x})$, which is a linear combination of features and constitutes the prediction of the desired output signal, given that the weights, w, are trained accordingly.

A basic form of an ESN is presented in Figure 1. The reservoir consists of a layer of K input units, connected to N reservoir units through an $N \times K$ weighted connection matrix W^{in}. The connection matrix of the reservoir, W, is an $N \times N$ matrix. Optionally, an $N \times L$ backprojection matrix W^{back} could

be employed, where L is the number of output units, connecting the outputs back to the reservoir neurons. Input units' (linear features) and reservoir units' (non-linear features) weights are defined in an $L \times (K + N)$ matrix, W^{out}.

Based on existing literature, the following rules should hold when creating ESNs and the respective matrices W^{in}, W and W^{back}. Briefly, these are: (i) W should be sparse, (ii) the mean value of weights should be around zero, (iii) N should be large enough to introduce more features for better prediction performance, (iv) the spectral radius, ρ, of W should be ≤ 1 to practically (and not theoretically) ensure that the network will be able to function as an ESN. Finally, a weak uniform white noise term can be added to the features for stability reasons. A detailed discussion on ESNs best practices can be found in [6].

In this work, we consider discrete time models and ESNs without backprojection connections. In addition, for the current implementation, the reservoir units employ $f(x) = tanh(x)$ as an activation function, while the output units employ the sigmoid function $g(x) = \frac{1}{1+e^{a \cdot x}}$. We scale and shift the input signal, $\mathbf{u} \in \mathbb{R}^K$, depending on whether we want the network to work in the linear or the non-linear part of the sigmoid function. The reservoir feature vector, $\mathbf{x} \in \mathbb{R}^N$, is given by Equation 1:

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(t+1) + \mathbf{W}\mathbf{x}(t) + \mathbf{v}(t+1)) \tag{1}$$

where $\mathbf{f}$ is the element-wise application of the reservoir activation function and $\mathbf{v}$ is a uniform white noise vector. The output, $\mathbf{y} \in \mathbb{R}^L$, is then given by Equation 2:

$$\mathbf{y}(t+1) = \mathbf{g}(\mathbf{W}^{out}[\mathbf{u}(t+1)|\mathbf{x}(t+1)]) \tag{2}$$

where $\mathbf{g}$ is the element-wise application of the output activation function and $|$, the aggregation of vectors.

For RL tasks with K continuous states and L discrete actions, we can use an ESN to model a Q-value function, where each network output unit l, can be mapped to an action $a_l, l = 1 \ldots L$, with the network output value y_l denoting the long-term discounted value, $Q(\mathbf{s}, a_l)$ of performing action a_l, when the agent is at state $\mathbf{s}$. Given $g(x) = x$, this Q-value can be represented by an ESN as:

$$y_l = Q(\mathbf{s}, a_l) = \sum_{i=1}^{K} w_{li}^{out} s_i + \sum_{i=K+1}^{K+N} w_{li}^{out} x_{i-K}, l = 1, \ldots, L \tag{3}$$

while actions can be chosen under the ϵ-greedy policy [3]. Linear Gradient Descent (GD) SARSA TD-learning can be used to adapt weights [3,7] but in this work the weights will be perturbed through evolutionary computation as a form of policy search.

2.3 NeuroEvolution of Augmented Reservoirs

NeuroEvolution of Augmented Topologies (NEAT) [8] is a topology and weight evolution algorithm for artificial neural networks, founded on four principles.

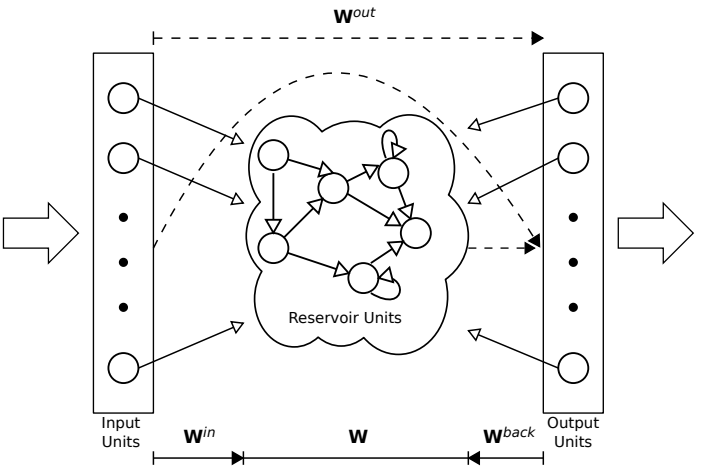


Fig. 1. A basic form of an ESN. Solid arrows represent fixed weights and dashed arrows adaptable weights.

First, the network, i.e. the phenotype, is encoded as a linear genome (genotype), making it memory efficient with respect to algorithms that work with full weight connection matrices. Secondly, NEAT employs the notion of *historical markings*, in order to annotate with innovation numbers, newly created connections. During crossover, NEAT aligns parent genomes by matching the innovation numbers and performs crossover on these matching genes (connections). The third principle is to protect innovation through *speciation*. Organisms are clustered into species in order to have time to optimize by competing only in their own niche. Last but not least, NEAT starts with minimal networks (networks with no hidden units), in order start with a minimal search space and to justify every complexification made in terms of fitness. NEAT complexifies networks through the application of structural mutations (adding nodes and connections), and further adapts the networks through weight mutation (perturbing or restarting weight values). These primitives can be employed in other NE settings also, in the form of a meta-search evolutionary procedure. In our case, we consider them to achieve efficient search in the space of ESNs.

NeuroEvolution of Augmented Reservoirs (NEAR) [9] further utilizes NEAT as a meta-search algorithm and adapts its four principles to the ESN model of neural networks. NEAR adapts the NEAT search algorithm mainly with respect to gene representation, crossover with historical markings, clustering and including some additional evolutionary operators related to ESNs. An important difference from NEAT is that both evolution and classic learning algorithms (least squares, temporal difference learning) can be employed in order to adapt networks to the problem at hand.

3 Developed Mechanism

The mechanism built aims to help the agent select the RFQs to bid on and the offers to make. Apart from the reserve price of the RFQs (more in TAC SCM specifications), the agent is constrained also from component availability and daily factory capacity. Nevertheless, focus of the current work is on optimizing the selling and factory utilization of the agent. To this end, two basic assumptions have been made: a) the agent has an infinite, on-time, availability of components, decoupling any procurement modules from our study and b) all algorithms will bid on the current RFQs, without holding any cycles for future orders. In competition mode, by generating predicted RFQs for future days [10] and adding them to the current RFQ portfolio, the algorithms will implicitly save the cycles by including fake, but more profitable RFQs.

The agent's bidding portfolio employs the utility maximization approach [10,1], due to the uncertainty imposed by the presence of competitors. Since, procurement strategies are ignored and thus costs minimization is not taken into account, utility is calculated based on revenue, rather than profit, without making any comparison between bidding mechanisms unfair. The RFQs are sorted based on the estimated utility, normalized over the product's required manufacturing cycles [11]:

$$Utility = \frac{P(offer = accepted|bid) \cdot BidPrice}{Cycles(Product)}. \quad (4)$$

In the developed mechanism, we have adopted the *partial order approach*, where we assume that we win a part of the order, proportional to the probability of acceptance at the given bid price [10]:

$$PartialCycles = P(offer = accepted|bid) \cdot Quantity \cdot Cycles(Product). \quad (5)$$

Even if one cannot win partial orders, the idea is that in the end, the total quantity of cycles won would be close to the total quantity of cycles won, if partial orders were available in the game.

The selected RFQs are the set:

$$SelectedRFQs = \{x \in RFQBundle| \sum PartialCycles(x) \leq 2000; \text{ and } Utility(x) > Utility(y), y \notin SelectedRFQs\}. \quad (6)$$

Algorithm 1 summarizes the above methodology.

3.1 Probability Estimation

On the problem of estimating the probability of acceptance of an offer given a price a lot of work has been done. Other works have approached the same problem using linear regression [11,12], particle filters [10], heuristics [13], sigmoid curves [14] or even more elaborate schemes with a combination of radial basis

Algorithm 1. Utility maximization given bidding prices. These bidding prices can be winning prices predicted from past games, or prices derived by some optimization procedure.

```
For each RFQ select a price to bid on
For each RFQ bid calculate the probability of acceptance
For each RFQ calculate its estimated utility
Sort RFQ Bundle based on utility
CyclesRemaining ← 2000
while CyclesRemaining > 0 do
    RFQ ← RFQBundle.next()
    CyclesRemaining− = RFQ.quantity · RFQ.prob · RFQ.cycles
    OfferBundle.add(RFQ)
end while
Send offers
```

function networks, economic regimes and on-line correction mechanisms [15]. In our case, we used the logistic regression learning algorithm, whose parametric nature provides us with the mechanism of adjusting a derived sigmoid curve to match current RFQ properties and market conditions in the form of features. The probability of acceptance is given by the equation:

$$f(z) = \frac{1}{1 + exp^{-z}}, \tag{7}$$

where

$$z = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n, \tag{8}$$

and x_i denotes the corresponding feature. Optimal weights are found through optimization routines. If gradient descent is used, the logistic regression model has the advantage of handling vast amounts of data or even adapting on-line through iterative weight updates.

Nine features are utilized for parameterizing the logistic regression curve. Features can either be properties of the RFQ or properties of the market conditions. The RFQ related features are: 1) *current date* (CD) 2) *due date* (DD) 3) *reserve price* (RP) 4) *quantity* (QNT) and 5) *product base price* (BP). On the other hand, market related features are: 6) *5-day average maximum selling product price* (AMAX) 7) *5-day average minimum selling product price* (AMIN) and 8) *total quantity of customer requested products* (TQNT). Finally we have, 9) *offer price* (OP). The target variable is whether the offer at the specific price, denoted by the ninth feature, was accepted (1) or not (0). Each feature was normalized according to Equation 9. When available, the minimum and maximum prices are taken from the game specifications, otherwise the values are derived from the training data, where due to the vast amount of available samples, the range includes any test sample presented to the model.

$$x' = \frac{x - \bar{x}}{range(x)} = \frac{x - \bar{x}}{max(x) - min(x)} \tag{9}$$

The final probability is given by:

$$Pr(accepted|bid) = \begin{cases} f & \text{if } bid <= \text{reserve price,} \\ 0 & \text{if } bid > \text{reserve price.} \end{cases} \quad (10)$$

3.2 Price Estimation

The second component of the developed mechanism is responsible for estimating the best price to offer. This problem can be approached either by using stochastic optimization [16] or greedy search [12]. In both cases the idea is that given an acceptance probability estimation, we search for the prices that will optimize the offering bundle under the given constraints. The former approach requires specialized software and large computational power for deriving near real-time solutions, so we turn out focus on the latter approach.

Winning Price Prediction. The first approach is to employ data mining on historical data in order to predict winning prices; classification and regression trees (CART) [17] or M5' [18] are typical algorithms for doing so. Using as predictor variables the features 1 through 8 described in Section 3.1, our goal is to predict the price that will convert our offer into an order. Previous work following this approach can be found in [1] using M5' and in [19] using k-Nearest Neighbors. The major problem with this approach is that the models are fitted on log files from games with other agents than those in operational mode. And, even though features related to market conditions are informative, the non-stationarity of the process leads to a decrease in prediction performance. For this reason the predictions are often corrected on-line [19,15].

Particle Swarm Optimization. Particle swarm optimization (PSO) [20] has also been applied. Our PSO approach was designed so that each individual represents a solution, i.e. contains a vector of prices for every RFQ in the RFQ bundle. Initial values are sampled randomly over the space $[0.9p, 1.1p]$, where p is the price predicted by M5' implementation (Section 3.2). Associated with each particle, besides the vector of prices $\tilde{x}$, is a velocity vector $\tilde{v}$ and a fitness value f. Each individual has a memory storing the highest fitness solution it has experienced since the beginning of the algorithm, denoted as P_{bst}. In addition, each particle, is also aware of the fittest individual in its neighborhood, G_{bst} (in this variant of PSO, the neighborhood is defined as the entire population).

In every iteration the particles update their "positions", $\tilde{x}$, based on their "velocities", whereas the velocities of each particle are based on its P_{bst} and the G_{bst} vectors. Under this policy, the particles steer towards the fittest particle in the population and the fittest region of the space each has visited so far. The state transition equation is thus:

$$\tilde{v}_t = \chi(\tilde{v}_{t-1} + r_1\phi_1(P_{bst} - \tilde{x}_{t-1}) + r_2\phi_2(G_{bst} - \tilde{x}_{t-1})) \quad (11)$$

$$\tilde{x}_t = \tilde{x}_{t-1} + \tilde{v}_t \quad (12)$$

where χ is the inertial coefficient, ϕ_1 and ϕ_2 are acceleration constants and r_1, r_2 are sampled randomly from $U(0,1)$. The inertial coefficient defines how reactive a particle is to velocity changes, the acceleration constants define the factor by which particles tend to fly toward their P_{bst} and G_{bst} positions, and the random variables are used to introduce stochasticity into the transition. Stochasticity is necessary to jostle particles out of an equilibrium that occurs at points between P_{bst} and G_{bst} which are not necessarily optimal [20]. PSO has an advantage over genetic algorithms with respect to its relative finesse with real values.

Bidding via Control. An alternative approach is to view the problem as a control problem. The goal is to adjust prices offered, in order keep a metric near its optimal value. For example this metric could be related to having 100% factory utilization without any late or canceled orders. If the prices proposed by the agent are high, then the factory utilization will drop as a consequence of not winning any orders. The opposite holds for high factory utilization and missed or delayed orders. Previous approaches in the TAC SCM domain include distributed feedback control, fuzzy logic control or heuristic control, respectively [21,22,13].

In our proposed mechanism, we have employed the heuristic control method introduced by `PhantAgent` [13], winner of the 2007 competition, and posed it as a reinforcement learning problem. Our goal is to find an appropriate policy that will optimize the agent's selling and manufacturing performance through adaptive function approximation. `PhantAgent`'s control strategy is presented in Algorithm 2. Based on how many orders the agent wins, the utilization of its factory and the orders queued for production, `PhantAgent` manipulates a factor f that adjusts the 3-day maximum of the maximum selling price for the specific product in the RFQ. We conjecture that its limitation is the adaptation strategy of factor f, which is not optimized and thus performance gains can be achieved.

Algorithm 2, slightly modified, could be posed as a reinforcement learning (RL) control problem. The Markov decision process is formed as follows:

$$S = \{won\ cycles/capacity, queued\ cycles/capacity, total\ quantity/3500\} \quad (13)$$

$$A = \{0.9, 0.91, 0.92, ..., 1.04, 1.05\}, |A| = 16 \quad (14)$$

$$r = -|\frac{won - capacity}{capacity}| - |\frac{queue - capacity}{capacity}| \quad (15)$$

The engineering of the reward function was made in order to punish a bidding strategy that would diverge from winning 2000 cycles a day and have 2000 cycles queued each day in the factory, waiting to be manufactured. The state variable *total quantity* was normalized between 0 and 1 using an upper bound of 3500 PCs requested over one day by the customers. The bound can be estimated both from the game specifications and from the training data.

4 Experiments and Results

Experiments were conducted using log files from the TAC SCM 2011 finals (16 games) and semi-finals (20 games). All the models (logistic regression, CART,

Algorithm 2. PhantAgent's heuristic control algorithm. The algorithm keeps track of several variables: factor f with initial value 1, variable *won*, which holds the total number of cycles needed to assemble all the bids won in the last day, *cap*, the factory capacity for one day, *price*, the offer price, *high*, the highest of the last 3 days highest winning prices and *pqueue*, the total number of cycles needed to assemble all the computers that still have to be delivered

$f = f + \frac{(won - cap)}{(cap * 5)}$
if $pqueue > (2 * cap)$ **then**
  $f \leftarrow f + 0.01$
end if
if $pqueue < cap$ **then**
  $f \leftarrow f - 0.005$
end if
if $f < 0.9$ **then**
  $f \leftarrow 0.9$
end if
if $f > 1.05$ **then**
  $f \leftarrow 1.05$
end if
$price \leftarrow high \cdot f$

M5' and NEAR) were trained on semi-finals data and tested with the finals logs. The supervised learning schemes were trained with 40% of the semi-finals data, corresponding to 952953 tuples for the logistic regression (includes both accepted and non-accepted offers) and 312891 tuples for the regression tree methods (includes only accepted, won, offers). The R package `rpart` [23,24] and the WEKA library [25] were employed for the CART implementation and the M5' implementation respectively. Simulations were performed on the finals games. We have kept only RFQs for which at least one offer was made, in order for the final results to be in accordance with competition mode.

First we tested the logistic regression fit. Feature weights are provided in Table 1. Figure 2 illustrates the fitted logistic curves for a specific RFQ over the range of possible offer price values $[0, 1.25 \cdot basePrice]$, for different values for reserve price (Figure 2a) and due date (Figure 2b). One may easily identify their effect on the curve. A quantitative metric of the logistic regression performance is to predict whether an order will be accepted ($Pr \geq 0.5$) or not ($Pr < 0.5$). CART predicts a bidding price for every unseen finals' RFQ and logistic regression predicts the probability of acceptance. The results provide a 72.6% classification accuracy.

Selling and throughput factory performance of the different approaches was performed on the adjusted total revenue (adj.tot.rev.) value, which follows the equation:

$$\text{adj.tot.rev.} = \text{tot.rev.} - max(\overline{\text{daily cycles}} - 2000, 0) \cdot \overline{\text{lost cycle cost}}. \quad (16)$$

Table 1. The weights of the logistic regression fit. The most important features are the offer price (`OP`) and the average maximum report price (`AMAX`), having negative and positive effect respectively on the probability of a successful bid the higher they are. Third in importance is the reserve price (RP). Importance is based on their contribution to the final outcome and is comparable due to normalization performed before fitting.

CD	BP	DD	QNT	AMAX	AMIN	TQNT	RP	OP
-0.19	-0.67	-1.26	0.80	17.96	1.61	0.52	2.39	-22.94

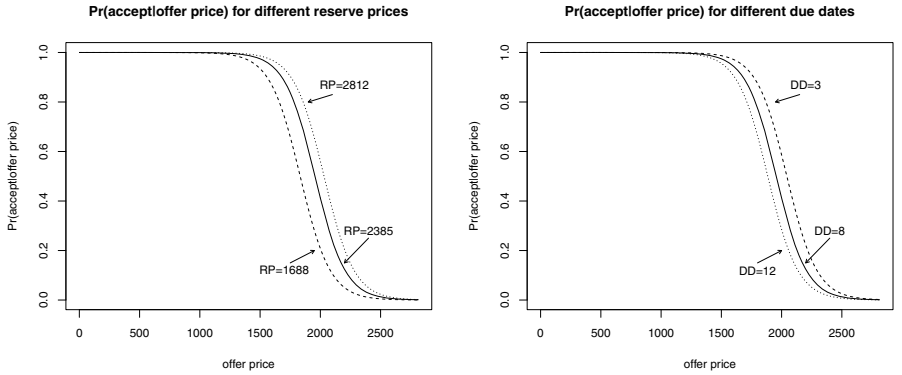
Equation 16 adjusts the total revenue (tot.rev.) to include costs incurred by the extra average daily cycles (daily cycles) won above the 2000 capacity limit. In order to account for the extra cycles and make the results comparable, we identify how much an extra cycle costs to the agent on average (lost cycle cost), in order to subtract it from the total revenue and compare the results again. The average RFQ has 10 items to deliver with 5.5 average manufacturing cycles per item, summing up to 55 cycles. The average price of 10 items is 10 times the average base price (2000), equal to 20000. The penalty incurred is on average 10%, thus the total price we subtract from the revenue is $20000 \cdot 1.1 = 22000$ for 55 cycles (that it 400 per cycle). If this is true for 216 days, then the amount to subtract for every extra average cycle is 86400. For example, if we have 100 extra average cycles, then $86400 \cdot 100 = 8640000$ must be subtracted in the end from the total revenue.

NEAR experiments were executed with a population of 100 networks, for 100 generations. The performance of the champion in terms of average total reward over all the finals games behavior is depicted in Figure 3 and is indicative of the evolutionary pressure applied by the algorithm to discover efficient networks.

PSO included 100 particles which were allowed to iterate 100 times. At a standard desktop, it took around $1''$ of processing time. Thus, this setup allows adequate time to add predicted RFQs in order to have a 10 day optimization plan when in competition. PSO parameters χ and $\phi_{1,2}$ were set to 0.1 and 0.5, respectively.

Table 2 summarizes the adjusted total revenue (ATR) and the average daily cycles (ADC) won for each one of the 2011 finals games, for all the techniques presented.

Based on the guidelines in [26] we employed the Wilcoxon signed-ranks test [27] for pairwise statistical significance comparison between the RL method and the rest. We calculate the statistic z and if $z < -1.96$ we can discard the null hypothesis that the two competitors have the same performance on average for $\alpha = 0.05$. The test gave values $z = -2.32$, $z = -2.53$, $z = -2.63$ and $z = -3.51$ when comparing RL versus CART, M5', PSO and heuristic control respectively, providing statistical confidence over the superiority of the RL/NEAR method in pairwise comparisons.



(a) Different reserve prices (b) Different due dates

Fig. 2. Both figures display the probability of accepting an offer (y-axis) given the offer price (x-axis) for different values of other variables that affect the curves like due date and reserve price. The representation of the acceptance probability through a sigmoid function is close to the reality, since prices near the reserve price are unlikely to be accepted in a competitive environment, while low prices will be accepted almost certainly. The steep descent should be at the price the current market condition indicate.

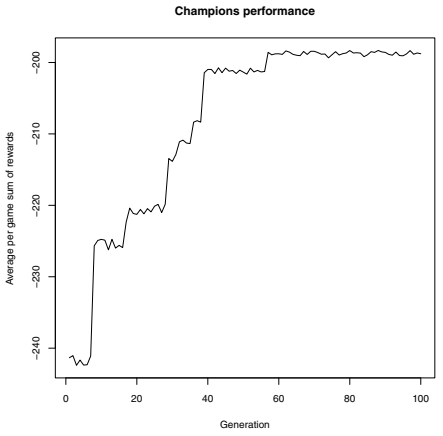


Fig. 3. The average per game (episode) sum of rewards gathered over the 216 days of the champion network per generation. We can observe how the evolutionary pressure drives the policy search towards more efficient networks.

Table 2. Adjusted total revenue (ATR) and average daily cycles (ADC) for all finals games. ATR is in millions.

Game ID	CART		M5'		Heuristic		PSO		RL	
	ATR	ADC	ATR	ADC	ATR	ADC	ATR	ADC	ATR	ADC
4584	119.9	2338	121.1	2256	124.9	1882	120.7	2287	**129.2**	1984
4585	**126.2**	2174	123.9	2157	121.5	1909	123.5	2181	125.6	1971
4586	123.0	2259	122.4	2130	121.8	1915	121.4	2134	**125.7**	1994
4587	**132.0**	2321	125.6	2294	125.4	1930	125.4	2322	128.9	1995
4588	123.4	2324	126.1	2252	125.0	1908	124.8	2302	**129.4**	1992
4589	123.6	2289	123.4	2251	124.0	1906	122.9	2258	**130.0**	1996
4590	108.2	2239	108.9	2089	104.5	1827	108.0	2141	**109.6**	1910
4591	**135.0**	2084	132.1	2003	124.5	1831	132.9	2059	128.3	1906
4360	137.6	2234	132.0	2206	141.4	1921	135.3	2181	**145.1**	1990
4361	120.8	2184	119.4	2116	119.4	1862	120.3	2113	**123.6**	1944
4362	112.7	2290	111.0	2088	112.9	1905	110.9	2130	**116.6**	1979
4363	97.8	2349	110.3	2054	109.5	1920	108.4	2107	**111.8**	1980
4364	116.0	2137	**119.1**	2100	107.8	1799	117.6	2143	113.1	1907
4365	120.4	2218	120.2	2120	124.8	1918	120.2	2125	**127.1**	1983
4366	118.9	2244	118.3	2049	118.9	1915	119.2	2098	**123.3**	1998
4367	97.1	2300	102.9	2102	105.9	1891	101.3	2155	**108.7**	1973
Average	119.5	2249	119.8	2141	119.5	1889	119.6	2171	**123.5**	1968

Besides heuristic control and RL, all the other methods continuously underestimate the winning price during simulations. As a consequence, they win more orders and have a factory utilization above the 2000 cycles limit. This behavior may seem strange at the beginning, since it is expected that the finals are more competitive and thus have lower market prices than the semi-finals (training data). The cause of this paradox is the specific mix of agents participating in the finals, which led to a less competitive set of games, evident by the range of the final bank accounts of 1st and 6th place, being $12M$ and $13M$ in 2009 and 2010 respectively, and $29M$ in 2011[1]. On the other hand, the control methods, and especially RL, are working only by receiving feedback from the current market conditions and agent state leading to better robustness and generalization behavior. Prior to experimentation. one could conjecture that the PSO appears to be a more efficient approach, since it tests a lot of solutions, picking the best one among many. The fact that it lags behind is probably due to the noisy premises induced by the probability estimation upon which the PSO method relies heavily on.

5 Conclusions and Future Work

In this work we have designed a bidding mechanism in the context of TAC SCM formed by three decoupled components: price selection, acceptance probability

[1] Results are taken from `http://tac.cs.umn.edu/`

estimation and offer selection. Under this scheme, performance boosts in any of the three components would increase profitability in the competition. In our case we have used logistic regression as our acceptance probability model, greedy utility maximization as our offer selection mechanism and experimented with five price selection mechanisms. Our adaptive function approximation methodology via policy search for controlling the bidding prices provided good generalization behavior on unseen market conditions and picked up as the best performing solution among the rest. In the future we plan to test the derived methodology of price control through policy search in this year's TAC SCM competition and continue to improve the logistic regression component by adding for example penalty terms on the weights during the optimization procedure.

References

1. Chatzidimitriou, K.C., Symeonidis, A.L., Kontogounis, I., Mitkas, P.A.: Agent mertacor: A robust design for dealing with uncertainty and variation in scm environments. Expert Systems with Applications 35(3), 591–603 (2008) (Cited by: Collins2008)
2. Stone, P.: Learning and multiagent reasoning for autonomous agents. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 13–30 (January 2007)
3. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
4. Arunachalam, R., Sadeh, N.M.: The supply chain trading agent competition. Electronic Commerce Research and Applications 4(1), 66–84 (2005)
5. Collins, J., Arunachalam, R., Sadeh, N., Eriksson, J., Finne, N., Janson, S.: The supply chain management game for the 2007 trading agent competition. Technical Report CMU-ISRI-07-100, Carnegie Mellon University (December 2006)
6. Jaeger, H.: Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach. Technical Report GMD Report 159, German National Research Center for Information Technology (2002)
7. Szita, I., Gyenes, V., Lőrincz, A.: Reinforcement learning with echo state networks. In: Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E. (eds.) ICANN 2006. LNCS, vol. 4131, pp. 830–839. Springer, Heidelberg (2006)
8. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2), 99–127 (2002)
9. Chatzidimitriou, K.C., Mitkas, P.A.: A neat way for evolving echo state networks. In: European Conference on Artificial Intelligence. IOS Press (August 2010)
10. Pardoe, D., Stone, P.: An autonomous agent for supply chain management. In: Adomavicius, G., Gupta, A. (eds.) Handbooks in Information Systems Series: Business Computing, vol. 3, pp. 141–172. Emerald Group (2009)
11. Benisch, M., Greenwald, A., Grypari, I., Lederman, R., Naroditskiy, V., Tschantz, M.: Botticelli: A supply chain management agent designed to optimize under uncertainty. ACM Transactions on Computational Logic 4(3), 29–37 (2004)
12. Pardoe, D., Stone, P.: Bidding for customer orders in TAC SCM. In: Faratin, P., Rodríguez-Aguilar, J.-A. (eds.) AMEC 2004. LNCS (LNAI), vol. 3435, pp. 143–157. Springer, Heidelberg (2006)

13. Stan, M., Stan, B., Florea, A.M.: A dynamic strategy agent for supply chain management. In: Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 227–232 (2006)
14. Chatzidimitriou, K.C., Symeonidis, A.L.: Data-mining-enhanced agents in dynamic supply-chain-management environments. Intelligent Systems 24(3), 54–63 (2009); Special issue on Agents and Data Mining
15. Hogenboom, A., Ketter, W., van Dalen, J., Kaymak, U., Collins, J., Gupta, A.: Product pricing in TAC SCM using adaptive real-time probability of acceptance estimations based on economic regimes. In: Workshop: Trading Agent Design and Analysis (TADA) at Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009), 15–24 (July 2009)
16. Benisch, M., Greenwald, A., Naroditskiy, V., Tschantz, M.C.: A stochastic programming approach to scheduling in TAC SCM. In: Proceedings of the 5th ACM Conference on Electronic Commerce, EC 2004, pp. 152–159. ACM, New York (2004)
17. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.: Classification and Regression Trees. Chapman and Hall (1984)
18. Wang, Y., Witten, I.H.: Induction of model trees for predicting continuous classes. Poster Papers of the 9th European Conference on Machine Learning, pp. 128–137 (1997)
19. Kiekintveld, C., Miller, J., Jordan, P.R., Callender, L.F., Wellman, M.P.: Forecasting market prices in a supply chain game. Electronic Commerce Research and Applications 8, 63–77 (2009)
20. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the International Conference on Neural Networks, pp. 1942–1948 (1995)
21. Kiekintveld, C., Wellman, M.P., Singh, S., Estelle, J., Vorobeychik, Y., Soni, V., Rudary, M.: Distributed feedback control for decision making on supply chains. In: Fourteenth International Conference on Automated Planning and Scheduling (2004)
22. He, M., Rogers, A., Luo, X., Jennings, N.R.: Designing a successful trading agent for supply chain management. In: Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2006 (2006)
23. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2011) ISBN 3-900051-07-0
24. Therneau, T.M., port by Brian Ripley, B.A.R.: rpart: Recursive Partitioning, R package version 3.1-50 (2011)
25. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explorations 11(1), 10–18 (2009)
26. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7, 1–30 (2006)
27. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin 1(6), 80–83 (1945)