www.icgst.com

Fuzzy Sarsa: An approach to linear function approximation in reinforcement learning

L. Tokarchuk, J. Bigham, and L. Cuthbert
Electronic Engineering, Queen Mary, University of London,
Mile End Road, London, E1 4NS
[l.n.tokarchuk,j.bigham, laurie.cuthbert]@elec.qmul.ac.uk,
http://www.elec.qmul.ac.uk

Abstract

This paper investigates two different approaches to learning using an agent electronic marketplace as test bed. The types of learning considered in this paper include the temporal difference (TD) learning algorithm Sarsa, and two new fuzzified versions of this algorithm, FQ Sarsa and Fuzzy Sarsa. We implement the three learning algorithms in an agent test bed in order to determine their usefulness in the context of an electronic marketplace. We present the results of various tests demonstrating that the Fuzzy Sarsa algorithm, while having the smallest state space, is also the more effective method of learning.

Keywords: *Fuzzy logic, Reinforcement Learning, Agent Systems.*

1. Introduction

Over the last few years there has been continued interest in developing systems based on autonomous agents. These systems often have complex and dynamic environments where fixed or stationary strategy agents have difficulty competing because of their inflexibility. Furthermore, fixed strategy agents become increasingly difficult to manage as the range of required agent behaviors multiplies. Therefore, it has become important that agents are capable of learning about and adapting to their environment. Learning agents reason about their surroundings by utilizing an online learning method such as one of those offered by the family of algorithms available in reinforcement learning [1]. These types of agents typically do better and need less supervision than those that do not learn.

A further specialization of self adapting agents is presented in the areas of agent coevolution and agent modelling. These areas focus on the possibility that a learning agent may be able to gain added benefit from observing and thus predicting the behaviour of the other agents in its environment. There have been several attempts to address this problem including methods based on game theory, such as the recursive modelling method algorithm (RMM) [2] and Vidal's extended framework [3]. Both these methods make the assumption of knowledge of other's value functions, and are also computationally very complex. Modelling techniques based on an agent's knowledge of another agent's value functions are difficult in many competitive environments. One approach that does make this assumption is a method of modelling suggested by genetics, namely coevolution. [4]. Coevolution allows a learner to evolve in response to new information. In other words, if a learner in some environment changes its strategy, a coevolutonary learner would observe that change, and evolve in response to it.

Reinforcement learning does not immediately present a solution for providing agent coevolution, as the size of the state space an agent must reason about becomes computationally too large in even moderately sized problem domains. Specifically there has been significant work using linear functional approximation [1], particularly in association with Tile Coding [5]. Another possible approach to the state space explosion problem is favoured by the control systems community. This approach is derived from fuzzy set theory. A fuzzy set is a mapping from a set of real numbers to a set of symbolic labels. A fuzzy state consists of a set of symbolic labels, to which a discrete number can be mapped. The basic principle of fuzzifying reinforcement learning is to utilise fuzzy sets in state representation. In this manner, we can represent many states with only a few fuzzy states. There have been several algorithms that utilise this idea and are presented as fuzzy reinforcement learning [6, 7, 8]. This approach has reported favourable results, however algorithms presented are off-policy algorithms, which has been shown to diverge with function approximation [9].

This paper presents an analysis of two fuzzy temporal difference (TD) algorithms. It compares implementations of both fuzzy TD algorithms with their founding algorithm, the TD reinforcement learning algorithm Sarsa. The first part of this paper reviews the Sarsa algorithm and presents the two new fuzzy reinforcement learning algorithms; FQ-Sarsa, a fuzzy learning accelerator for

Sarsa learning, and Fuzzy Sarsa, a fuzzy reinforcement learner using the framework presented by Bonarini. Finally, we present the results of a variety of different tests conducted on the three algorithms in an agent marketplace testbed.

2. Learning Algorithms

In the following sections, we briefly present the three implemented learning algorithms, Sarsa, FQ Sarsa and Fuzzy Sarsa. We revisit the Sarsa algorithm, discussing state representation in using Fuzzy set theory terminology. We follow that by presenting FQ Sarsa, a reduced state space algorithm that utilizes only fuzzy states in its fuzzification. Finally, we supply the full fuzzy Sarsa algorithm, Fuzzy Sarsa, an on-policy fuzzy leaner which utilizes the full fuzzification of fuzzy states and actions.

Sarsa

The general principle of the Sarsa algorithm can be summarized best by its name: **S**tate, **A**ction, **R**eward, **S**tate, **A**ction. In Sarsa, an agent starts in a given state and executes an action. After the action, the agent transitions to a new state and receives a reward based on the value of that new state. A state consists of a set of discrete or crisp values that represent its current circumstance. Figure 1 illustrates potential states for a marketplace agent. In this case, these values are the amount of money left, and the number of auctions remaining.

State	Money_Left	Auctions_Left
S1	12	3
S2	5	1

Figure 1: State (Crisp State) Representation

An agent recognizes the state it is currently in (say state S1), and executes some action. This action causes a translation to another state. The Sarsa algorithm attempts to learn the value, or Q-Value, of a state-action pair-Q(s,a). For the example state S1 in Figure 1 there would be several entries corresponding to the state and all available actions from that state. If the available actions are bid 8, bid 6, and bid 4, our entries for S1 become:

State		Action
Money_Left	Auctions_Left	Bid
12	3	8
12	3	6
12	3	4

Figure 2: State action (crisp) pairs.

Executing an action from S1, results in the agent moving into a new state. As with all reinforcement style algorithms, there must be a trade off between exploration and exploitation. An exploratory action or exploiting action is chosen as a result of the current policy. One type of selection policy is the ε greedy selection policy. This selection policy operates on the simple guideline of choosing the most optimal action based on the current known rewards or Q-values for all possible actions. The agent chooses which action to take based on maximising its reward. For every selection there is some probability ε that rather than choosing the optimal greedy action, the algorithm will choose randomly to explore other actions in the hope that they may lead to a more optimal solution. After an agent has executed an action in a particular state, the agent receives a reward based on whether or not they have achieved their objective. Sarsa is an on-policy algorithm. This means that the learning occurs only from the experience of the policy actually being followed. An on-policy learner selects an action, receives a reward and observes the new state. Full details on the Sarsa algorithm are provided by Sutton and Barto[1].

FQ Sarsa

The FQ Sarsa algorithm is based on the Sarsa algorithm. Essentially, it reduces the state space by storing the state representation in fuzzy sets. In all other respects, it behaves exactly like the Sarsa. The algorithm does not consider fuzzy actions or goal states, leaving these in their original crisp representation and thus cannot be considered a truly fuzzy algorithm. In this approach, a crisp state s matches a set of fuzzy states and these fuzzy states are paired with crisp action values. To determine the fuzzy state, a mapping from the set of real numbers representing the current state to a set of symbolic state labels is created. Consider the world descriptor Money_Left from the states described Figure 1.

The value of Money_Left in a crisp state consists of a discrete number, say ML(x), $x \in Z = [0..15]$. However, in a fuzzy state, the same value x maps to one or more of the fuzzy labels associated with Money_Left = [Lots_Money, Little_Money]. X's degree of belonging to any particular fuzzy label is defined by the membership function (μ) associated with the fuzzy set Money_Left. So for example, the μ_{Money_Left} might be described as:

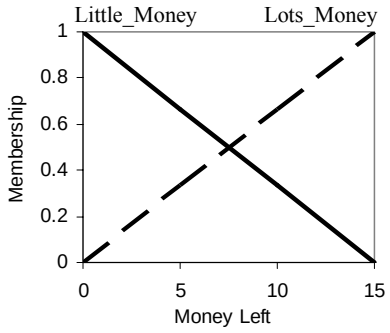


Figure 3: Membership function of Money_Left

The crisp values are then fuzzified using these membership functions. Each crisp value will belong to some degree, to one or more fuzzy set labels. In Figure 1, the fuzzification of $Money\ Left_{S1} = 12$ results in $\mu_{Lots\_Money}(12) = 0.87$ and $\mu_{Little\_Money}(12) = 0.13$. To fuzzify a crisp state, the membership of each state item is fuzzified, and, typically, the AND is calculated to obtain the state's membership or degree of matching. In the case of state S1 of Figure 1, crisp state S1 belongs to fuzzy states $\hat{S}1_b$ and $\hat{S}1_d$ with membership 0.87 and 0.13 respectively. All possible membership calculations for S1 are depicted in Figure 4.


Fuzzy State	Money Left	$\mu_{Money\ Left}$	Auctions Left	$\mu_{Auctions\ Left}$	μ_{S1}
$\hat{S}1_a$	Lots_Money	0.87	Few_Auctions	0	0
$\hat{S}1_b$	Lots_Money	0.87	Many_Auctions	1	0.87
$\hat{S}1_c$	Little_Money	0.13	Few_Auctions	0	0
$\hat{S}1_d$	Little_Money	0.13	Many_Auctions	1	0.13

Figure 4: Fuzzification of Crisp State S1

In FQ Sarsa, the actions are not fuzzified. As a result, the selection mechanism operates greedily rather than utilising any sort of fuzzy calculation mechanism, such as the centre of mass approach presented in the next algorithm. At any given time t, the action that is selected is the best action (the one with the highest FQ value) for the most fit fuzzy state (max $\mu(\hat{s}_t)$, where μ is the degree of matching of crisp state s to fuzzy state $\hat{s}$).

Rather than take the max of future rewards, we replace it with the FQ value of the new state action pair reached by applying the current policy - $FQ(\hat{s}_t, a_t)$. We choose a_t using the policy derived from FQ. In other words, $FQ(\hat{s}_t, a_t)$, is the state with the highest degree of matching ($max\ \mu(\hat{s}_t)$) and the action chosen follows the current policy (i.e. ε-greedy). We follow Berenji's example and take the fuzzy AND (or minimum) of $FQ(\hat{s}_t, a_t)$ and $\mu(\hat{s}_t)$.

```
All Q(s,a) values initialised (to 0 in our case).
Repeat for each episode (or auction game){
  Initialize ŝt (start state for the auction
  game).
  Choose at from ŝt using ε greedy policy.
  Repeat for each step(auction) in the
                  episode(auction game){
    Take action at, observe r and ŝt+1
    Choose at+1 from ŝt+1 using ε greedy policy
        FQ(ŝt-1, at-1) = FQ(ŝt-1, at-1) +
              α(r + λFQ(ŝt, at)^ μ(ŝt) − FQ(ŝt-1, at-1))
    ŝt = ŝt+1, at = at+1
  }
}
```

Figure 5: FQ Sarsa Algorithm

Fuzzy Sarsa

The FQ Sarsa algorithm presented above does not utilise fuzzy principles to combine actions, it only selects them. This approach is problematic in that essentially the FQ Sarsa algorithm only concentrates on reducing the state space and is not capable of fuzzy rule interaction. To that effect, we now examine the Fuzzy Q Learning algorithm presented by Bonarini and extend it in order to implement an on-policy learner.

Fuzzy Sarsa uses fuzzy representation of both states and actions. Its state/action entries do not include crisp actions like FQ Sarsa. Figure 6 illustrates the fully fuzzy state/ action pair used by the Fuzzy Sarsa algorithm. The degree of matching is still based on the fuzzy state, however membership functions for the fuzzification and defuzzification of fuzzy actions are now also required.

Fuzzy State		Fuzzy Action
Money Left	Auctions Left	Bid
Lots_Money	Many_Auctions	Bid_High
Lots_Money	Many_Auctions	Bid_Low

Figure 6: Fuzzy state action pairs

For example, Bid_High might defuzzify to the crisp action Bid 8. This type of fuzzy state action pair is referred to as a fuzzy rule where the fuzzy state corresponds to the antecedent of the rule and the fuzzy action proposed is the consequent. All fuzzy rules have a strength associated with them. It is this strength (FQ value) that most fuzzy reinforcement algorithms attempt to learn. In the action selection portion of a system, if a crisp state s matches a sub-populations' antecedent, the rule that is chosen is the rule with the highest strength value. However, since a crisp state s might match a number of fuzzy states (set $FS(s)$) as seen in Figure 4 (Both $\hat{S}1_b$ and $\hat{S}1_d$ match the fuzzy state S1), a method is needed in order to determine what action to take when all rules could be proposing different actions. For all $\hat{s} \in FS(s)$, there will be at *least* one matching fuzzy state action pair, or fuzzy rule (r). The action proposed for each $\hat{s}$, will be the greedy action (highest QS-value) proposed by the fuzzy rule. The final action proposed, is a weighted average of the actions proposed by each rule triggered. These actions are weighted in terms the degree of matching of the crisp state s, with the antecedent of the rule. The weighted average is computed using the centre of mass approach:

$$a = \frac{\sum_{i=1..n} \mu_i a_{\hat{s}_i}}{\sum_{i=1..n} \mu_i} \quad (1)$$

where n is the number of fuzzy states matching crisp state s, and $a_{\hat{s}_i}$ is the best action (having been de-fuzzified) proposed by any rule matching $\hat{s}_i$. Any fuzzy state with membership > 0 is considered in the action calculation.

μ	Fuzzy State		Fuzzy Action	FQ(ŝ,â)
	Money Left	Auctions Left	Bid	
0.7	Lots_Money	Many_Auctions	Bid_High	0.4
	Lots_Money	Many_Auctions	Bid_Low	0.1
0.4	Little_Money	Few_Auctions	Bid_High	0.2
	Little_Money	Few_Auctions	Bid_Low	0.6

Figure 7: Fuzzy state action pairs

Consider the crisp state s, which matches the two fuzzy states [Lots_Money, Many_Auctions] with degree 0.7 and [Little_Money, Few_Auctions] with degree 0.4. Each of these two fuzzy states has 2 rules associated with them. For the state [Lots_Money, Many_Auctions], the greedy action will be to Bid_High, since that rule has the highest FQ(ŝ, â) value. Similarly for the state [Little_Money, Few_Auctions], Bid_Low will be selected. The fuzzy actions are now defuzzified to obtain a crisp output. Bid_High is translated as bid 8 and

Bid_Low as bid 4. Thus the actual action taken is calculated as follows:

$$a = \frac{((0.7*8)+(0.4*4))}{(0.7+0.4)} = 6.5$$

For Fuzzy Sarsa, the Q value update formula is modified as follows:

$$FQ\ (\hat{s}^i_{t-1}, \hat{a}^i_{t-1}) = FQ(\hat{s}^i_{t-1}, \hat{a}^i_{t-1}) + \alpha\xi_{(\hat{s}^i_{t-1}, \hat{a}^i_{t-1})}(r_t + \gamma\sum_{\forall j} FQ(\hat{s}^j_t, \hat{a}^j_t)\xi_{(\hat{s}^j_t, \hat{a}^j_t)} - FQ_{t-1}(\hat{s}^i_{t-1}, \hat{a}^i_{t-1})) \quad (2)$$

where FQ values are the value of being in of fuzzy state and suggesting a fuzzy action, and $\xi_{c^i_{t-1}}$ is the fuzzification factor or the degree of belonging (μ) of the crisp state s_{t-1} to the fuzzy state $\hat{s}^i_{t-1}$. This is calculated as:

$$\xi_{(\hat{s}^i_{t-1}, \hat{a}^i_{t-1})} = \frac{\mu_{(\hat{s}^i_{t-1})}}{\sum_{i=1..n}\mu_i}. \quad (3)$$

We also have used $\hat{s}^i_{t-1}, \hat{a}^i_{t-1}$ rather than $\bar{r}$, since the current fuzzy state and suggested action is the definition of a fuzzy rule and r is already used in reference to the reward. In Q-learning, Q is updated using the largest possible reward (or reinforcement) from the next state, whereas in Sarsa, Q is updated with the value of the actual next state action pair as defined by the current policy. The change in the future contributions section to $\gamma\sum_{\forall i} FQ(\hat{s}^i, \hat{a}^i)\xi_i$ is again a result of the difference between Q-Learning and Sarsa. Rather than take the max of future rewards, we sum all rewards for all fuzzy states actions and multiply by the fuzzification factor. This is done for all FQ values where the fuzzy state $\hat{s}^j_t$ has some degree of matching to the next crisp state s, and the suggested action $\hat{a}^j_t$ is the action that would be applied using the current policy.

For these experiments we elected to ε-greedy action selection policy. However, it is not immediately clear how the algorithm should proceed in the exploratory case. In the crisp version of Sarsa, the exploratory action is chosen, say bid 8, and then the state action pair corresponding to the current state and bid 8 is used directly in learning. As discussed earlier, in fuzzy learning our crisp state matches *n* fuzzy states. Therefore, there are two possible ways of making an exploratory move. The first way, is that for each match made, a random move is generated and then the centre of mass of all the random moves is calculated to determine the actual action. The second way is to instead make a random move and consider the set of state/action pairs to be updated the set of *all matching* fuzzy state/action pairs ($\hat{s}_t$, $\hat{a}_t$), where $\hat{a}_t$ is the fuzzified crisp action, bid 8. Since we are trying to learn the specific action required with regards to the total set of matching fuzzy states, we elect to use the second method of exploratory action selection. Although only empirically tested, early experiments using both of these two methods indicated that the first method is tends cause instabilities in convergence. The remainder of fuzzy action selection is relatively straightforward: If a greedy action is taken, the algorithm observes the results and updates all fuzzy state/action pairs that contributed to the selection of $\hat{a}_t$. For example, in the random case if $\hat{s}1_a$, $\hat{s}1_b$ match our state and we randomly choose to do $\hat{a}_3$, the algorithm updates ($\hat{s}1_a$, $\hat{a}_3$),($\hat{s}1_b$, $\hat{a}_3$). If however, we chose the greedy action, then we would calculate the centre of mass of the actions proposed by $\hat{s}1_a$, $\hat{s}1_b$. Suppose $\hat{s}1_a$ proposed $\hat{a}_1$, and $\hat{s}1_b$ proposed $\hat{a}_3$ and that the centre of mass calculation returned $\hat{a}_2$. The pairs that are updated in the greedy case are the *contributing* pairs, ie. ($\hat{s}1_a$, $\hat{a}_1$)($\hat{s}1_b$, $\hat{a}_3$). After that is completed, the world is in a new state, and the algorithm repeats the above process for the new fuzzy state(s). The Fuzzy Sarsa algorithm [10] is presented as:

```
All Q(s,a) values initialised (to 0 in our case).
Repeat for each episode (or auction game){
  Initialize ŝt(start state for the auction game).
  Choose  ât  from  ŝt  by  calculating  the
    centre of mass using all ŝt that match
    crisp  s  and  ât  following  ε  greedy
    selection policy.
  Repeat  for  each  step(auction)  in  the
    episode(auction game){
      Take action ât, observe r and ŝt+1
      Choose ât+1 from ŝt+1 using ε greedy selection
      policy for all ŝt+1 match st+1.
      FQ(ŝ^i_{t-1}, â^i_{t-1}) = FQ(ŝ^i_{t-1}, â^i_{t-1}) +
                  αξ_{(ŝ^i_{t-1}, â^i_{t-1})}(r_t + γ Σ_{∀j} FQ(ŝ^j_t, â^j_t) ξ_{(ŝ^j_t, â^j_t)} − FQ_{t-1}(ŝ^i_{t-1}, â^i_{t-1}))
      ŝt = ŝt+1, ât = ât+1
    }
}
```

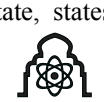
Figure 8: Fuzzy Sarsa Algorithm

3. An Agent Marketplace Test Bed

The marketplace used for this experiment is a first price sealed bid auction where the task for any agent is to win some number of items (*i*) during (*n*) auctions. Time is broken up into equal periods, each termed an episode. During each episode, a certain quantity *q* of items is available in *n* auctions. The algorithms that are implemented in this experiment attempt to learn a strategy over the episode of auctions. In the following discussion the terms *auction* and *auction game* have particular meaning. An auction refers to one event of auctioning an item within an episode, whereas an auction game refers to the set of auctions that take place during an episode.

In previously published work [10], we indicated that the algorithm Fuzzy Sarsa was the more powerful algorithm. However, this work did not go far enough in analysing the performance of the three learning algorithms. It only investigated the algorithms under a single learning situation. To determine whether Fuzzy Sarsa actually provides a better solution to Sarsa, we have implemented all three algorithms; Sarsa, FQ-Sarsa and Fuzzy Sarsa in an agent marketplace and conducted expanded tests with the three learning algorithms.

In the case of the Sarsa algorithm, we consider that the state of the world consists of 3 major categories: Money_Left, Auctions_Left and Items_Left. Actions included bids ranging from the offer price to the agent's maximum price and abstaining. Fuzzy States consisted of the same state categories as Sarsa. However, rather than storing the crisp representation of the state, states are

stored as fuzzy labels rather than discrete values. Unlike the work we previously presented, we elected to use three labels, since work currently being conducted on the efficacy of various different numbers of label combinations indicates that three labels is the most effective in this situation. Bonarini has indicated that additive membership functions, $\Sigma_{i=1 \text{ to } n}\ \mu_i(x) = 1$, are more robust under learning, thus the membership functions used here are triangular (rather than trapezoidal, etc), since triangular membership functions are popular and easy to design additive functions with. The general membership function used is given in Figure 9. This function is applied to the fuzzification of the state parameters [Money_Left, Auctions_Left, Items_Left] and, for Fuzzy Sarsa, Bid_Price.

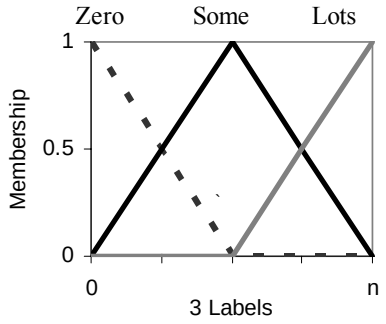


Figure 9: Fuzzy Membership Functions for the Test bed

In all tests conducted, to maximise exploration at the beginning of the simulation, an annealing factor is applied to ε and α until they reach a predefined minima (0.01). γ is set to 0.1 and the rewards used in this simulation were based on overall achievement of the agents' goal. They can be summarized as follows:

$$\text{POSITIVE REWARD} = I\left(\frac{M_t}{M_{t=0}}\right)$$

$$\text{NEGATIVE REWARD} = -I_{Needed}$$

$$\text{DEFAULT REWARD} = 0$$

Where I is the number of items, M is the amount of money at time t. The agent receives a positive reward at the end of the episode if it has achieved its goal (bought the required items) and a negative reward if it has not. At all other non-terminating state-actions, the agent receives the default reward. All tables are initialised to 0.

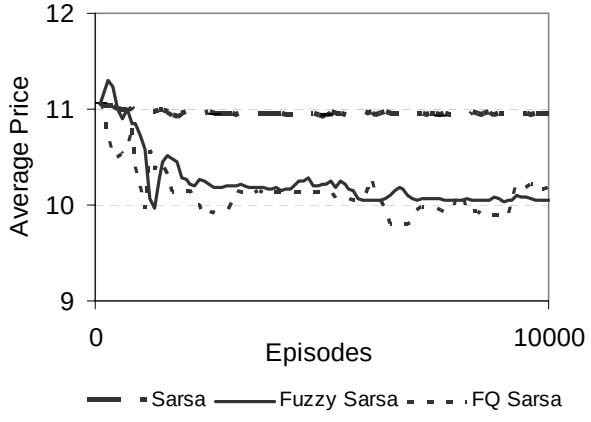


Figure 10: Large Game – Fixed Strategy Test

Since we altered the fuzzification parameters, our initial experiment which is a large scale test against a fixed strategy agent, is intended to determine how these changes affect the results presented in [10]. In this test, each agent is required to purchase 10 items from 20 auctions. Each agent is given enough money to purchase all 10 items at their maximum price. In all cases, the results presented represent an average over 10 games.

Figure 10 presents the results of our learning agents against a fixed strategy agent. As expected, the Fuzzy Sarsa agent finds a more optimal solution than either the FQ Sarsa Agent or the Sarsa Agent. In comparison with the results presented by Tokarchuk, the change in the number of fuzzification labels has indeed improved the performance of both the FQ Sarsa agent and the Fuzzy Sarsa agent. In fact, the FQ Sarsa agent seems to perform as well as the Fuzzy Sarsa agent. In order to determine the flexibility of the learning agents, they were next put into direct competition with each other. For this test, we kept all parameters (auction size, start money, rewards, etc) the same as in the first test. Each learning algorithm was tested against the other competitive algorithms in turn.

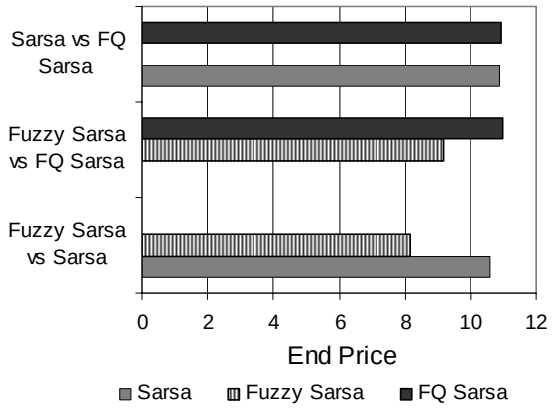


Figure 11: Large Game - Direct Competition Test

As shown in Figure 11, given the same learning parameters, Fuzzy Sarsa achieves a more optimal price than its competitors when in direct competition with either the Sarsa algorithm or the FQ Sarsa algorithm. The reason the Fuzzy Sarsa agent in the Fuzzy Sarsa vs. Sarsa game achieves a better price than that of the agent in the Fuzzy Sarsa vs. FQ Sarsa game is explained by Sarsa's inability to explore large state spaces sufficiently under these conditions. FQ Sarsa, because of its reduced state space, is more able to react to the Fuzzy Sarsa algorithm. In the Sarsa vs. FQ Sarsa test, it is interesting to note that Sarsa and its reduced state version, FQ Sarsa achieve an almost identical end price. While this result may initially seem encouraging for FQ Sarsa algorithm, more can be learned by examining the convergence data from the Sarsa vs. FQ Sarsa game as presented in Figure 12. Convergence, in this case, is measured as the ability of an agent to achieve the state goals of the game, to win 10 items from 20 auctions. Although Sarsa and FQ Sarsa both achieve equally good prices, FQ Sarsa is more prone to taking a risk with its bidding and thus failing to meet its stated objective.

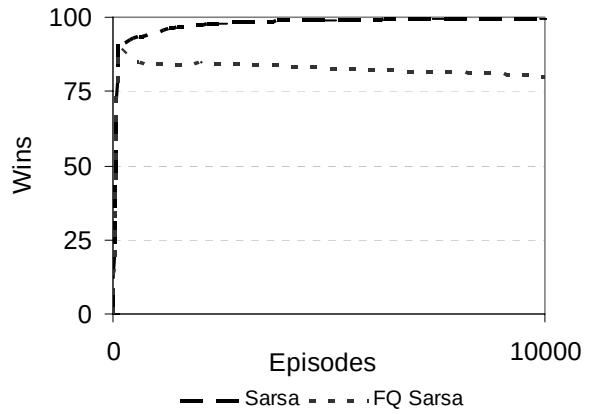


Figure 12: Large Game Convergence – Sarsa vs FQ Sarsa

In our final test, we put all 3 learning algorithms into the same game. In this test, each agent must still win 10 items, however in order to provide the same framework as the previous test, 30 auctions are now available.

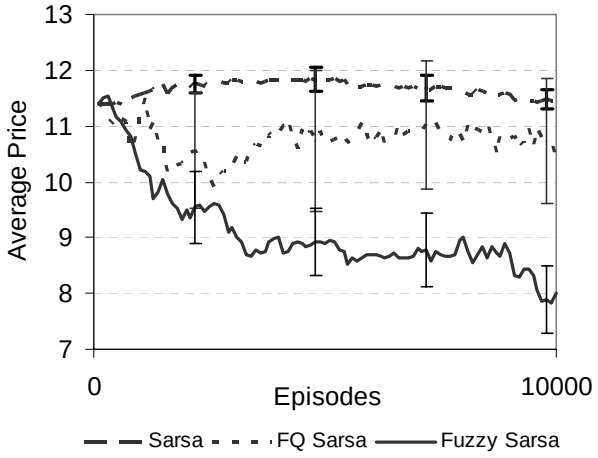


Figure 13: Sarsa vs FQ Sarsa vs Fuzzy Sarsa

In this final test, it is clear that Fuzzy Sarsa once again is the most optimal and flexible of the three algorithms. When competing directly with either of the other two algorithms it is able to consistently achieve a better price with minimal variability in end price. These results are significant because they demonstrate Fuzzy Sarsa's ability to learn effectively against a moving target, the other two learning agents. Under these settings the generalisation powers of Fuzzy Sarsa enable it to quickly take advantage of the current market conditions. Although not presented here for the sake of brevity, Fuzzy Sarsa experiences none of the convergence difficulties that FQ Sarsa encounters, as demonstrated in Figure 12.

4. Conclusions and Future Work

We have presented two fuzzy reinforcement learning algorithms. We have found that in relatively large scale state space, a pure fuzzy logic approach to reinforcement learning as presented in Fuzzy Sarsa allows for a more robust and correct solution than that presented by either Sarsa or than the reduced state space algorithm FQ Sarsa. This is a significant result since Fuzzy Sarsa works with a significantly smaller state space than Sarsa.

We intend to analyse this algorithm with respect to other forms of linear approximation algorithms specifically with SMDP Sarsa(λ) with linear tile-coding function approximation presented in [5]. After this has been completed we plan to test the algorithm against the tile coding approach in coevolutionary scenario.

The primary weakness we expect to see in future comparisons is a degradation of performance in comparison to tile coding in domains where the state space is described by many state variables. This degradation is expected because of the explosion in number of possible matching fuzzy state/action pairs. In tile coding, the number of features triggered stays the same irrespective of the number of state variables. However, since Fuzzy Sarsa generalises around membership boundaries, it is expected that it will be able to adapt to other learners in a coevolutionary scenario quicker than tile coding.

References

[1] Richard S. Sutton, Andrew Barto, *Reinforcement Learning: An Introduction*; MIT Press, Cambridge, MA, 1998.

[2] P. J. Gmytrasiewicz, E. H. Durfee, and D. K. Wehe. A decision-theoretic approach to coordinating multi-agent interations. *In Proc. Int. Joint Conf. on Artif. Intell.*, pages 62--68, 1991.

[3] J.M. Vidal and E.H. Durfee, Agents learning about agents: A framework and analysis; *In AAAI-97 Workshop on Multiagent Learning;* 1997.

[4] Melanie Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, 1998.

[5] Peter Stone, Richard S. Sutton, Gregory Kuhlmann, Reinforcement Learning for RoboCup-Soccer Keepaway, Adaptive Behavior, 2005.

[6] Hamid R Berenji, Fuzzy Q-Learning: A new approach for fuzzy dynamic programming, *IEEE World Congress on Computational Intelligence., Proceedings of the Third IEEE Conference on ,* 26-29 June 1994.

[7] Andrea Bonarini, Delayed Reinforcement, Fuzzy Q-Learning and Fuzzy Logic Controller, *In Herrera, F., Verdegay, J. L. (Eds.) Genetic Algorithms and Soft Computing, (Studies in Fuzziness, 8),* Physica-Verlag, Berlin, D, 447-466, 1996.

[8] Andrea Bonarini, Reinforcement distribution for fuzzy classifiers: a methodology to extend crisp algorithms, *Proceedings of the IEEE World congress on Computational Intelligence (WCCI) - Evolutionary Computation*, IEEE Computer Press; Piscataway, NJ; 51-56; 1998.

[9] Christopher J. C. J. Watkins, *Learning from Delayed Rewards,* PhD thesis, King's College, Cambridge, UK., 1989.

[10] L Tokarchuk, J Bigham, and L Cuthbert, Fuzzy Sarsa: An approach to fuzzifying Sarsa Learning, Proceedings of the International Conference on Computational Intelligence for Modeling, Control and Automation, 2004