Fuzzy Sarsa: An approach to fuzzifying Sarsa Learning

L. Tokarchuk[1], J. Bigham[1], and L. Cuthbert[1]

[1]Electronic Engineering, Queen Mary, University of London,
Mile End Road, London, E1 4NS

E-mail: {l.n.tokarchuk,j.bigham, laurie.cuthbert}@elec.qmul.ac.uk

Abstract

This paper discusses the two general approaches for learning in an agent electronic marketplace: reinforcement learning and fuzzy reinforcement learning. It examines the implementation of the TD learning algorithm Sarsa, before examining an approach to TD fuzzy reinforcement learning. We then extend the fuzzy learning algorithms in order to be comparable to the on-policy TD learning of Sarsa, FQ Sarsa and Fuzzy Sarsa. This paper discusses some of the issues around implementation of the reinforcement algorithm Sarsa, FQ Sarsa and Fuzzy Sarsa in an agent auction game. Finally, we discuss the convergence and optimality results of implementing these approaches to learning in an agent marketplace game.

1 Introduction

Over the last few years there has been a continued interest in developing systems based on autonomous agents. Given that interest, it is increasingly important that agents are capable of learning about the complex and dynamic environments that they exist in. A fixed or stationary strategy in such an environment rarely serves as the best policy for an agent. Agents that have the ability to learn about their surroundings typically do better and need less supervision than those that do not. As agent societies become more complex, agent behaviours become more difficult to manage. One technique of managing agent behaviours is to have self-adapting agents. These agents learn about their surroundings by utilising an online learning method such as one of those offered by the family of algorithms available in reinforcement learning [1]. This type of learning is appropriate in domains where existing data sets are not necessarily available for training.

One topic that has sparked interest in the last few years is the possibility that a learning agent may be able to gain added benefit from observing the other agents in its environment. Moreover, there are certain situations where agents may gain added benefit from being able to model and predict the behaviours of other agents they encounter. 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 framework [3]. RMM has several downsides. The first of these is the assumption of knowledge of other's value functions, while another is the computational complexity. Modelling techniques based on an agent's knowledge of another agent's value functions are inherently flawed in a competitive environment. Therefore, we need another method of modelling. One such method is suggested by genetics, is that of coevolution [4]. Coevolution allows a learner to evolve in response to new information. In other words, as learner one changes its strategy another learner changes to cope with it. Reinforcement learning does not immediately present a solution for providing agent co-evolution, as it suffers the problem of dimensional explosion. As the problem space in reinforcement learning becomes larger, so does the state space which

an agent must reason about. In order for an agent to evolve alongside other learning agents, that agent must maintain a model of the other agents. Since adding agent modelling to reinforcement learning would vastly increase the state space an agent needs to learn, we first investigate ways of decreasing the state space.

Fuzzy sets theory presents a possible solution to reducing the state space. 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 fuzzification is to utilise fuzzy sets in state representation. Therefore, 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 [5, 6].The first step we take is to represent our learning problem in terms of fuzzy logic. We call this process fuzzification of the learning problem. We then present a new version of the Sarsa [1] learning algorithm introduced by Sutton and Barto, called FQ Sarsa, which simply reduces the learning state space. We then investigate a more "fuzzy" approach by using the framework presented by Bonarini [7].

The paper is divided into three major sections: Section 2 describes the testing environment for our reinforcement algorithms. Section 3 discusses the Sarsa reinforcement learning technique, and how to apply it to a marketplace game. Section 4 discusses issues around fuzzifying marketplace state space. Section 5 presents our algorithms, FQ-Sarsa and Fuzzy Sarsa. Finally, Sections 6 and 7 present the results of implementing such algorithms in an agent marketplace.

2 An Auction Test Bed

In order to examine the reinforcement algorithms presented in the rest of this paper, a simple auction marketplace environment was simulated. The marketplace is a first price sealed bid auction where the task for any agent is to win some number of items (i) during (n) auctions. Essentially, time is broken up into equal periods, each termed an episode. An episode is comprised of the time taken to complete n auctions. During each episode , a certain quantity q of items is available. 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. The following sections present the reinforcement learning algorithms implemented in our marketplace; namely:

- Sarsa – a classic temporal difference (TD) learning reinforcement learning algorithm.
- FQ Sarsa – a fuzzy learning accelerator for Sarsa learning.
- Fuzzy Sarsa – our "fuzzification" of Sarsa following Bonarini's guidelines.

3 Sarsa

Sarsa is an on-policy TD learning algorithm. The general principle of Sarsa is summarized by its name: **S**tate, **A**ction, **R**eward, **S**tate, **A**ction. In Sarsa, an agent starts in a given state, from which it does some action. After the action, the agent receives a reward and has transitioned into a new state from which it can take another action. In reinforcement learning, a state consists of a set of discrete values representing the current state of the world. Figure 1 illustrates potential states for a marketplace agent. In this case, the agent has some discrete values which make up its state of the world. These are the amount of money left, and the number of items still left to buy.

State	Money_Left	Items_to_Buy
S1	12	3
S2	5	1

Figure 1: State (Crisp State) Representation

In Sarsa, the current state of the agent's environment is represented by a particular state. The agent recognizes which state it is in (say state S1), and executes some action. This action causes a translation to another state. In traditional reinforcement learning algorithms like Sarsa, we attempt 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 in our table corresponding to all the possible actions. If the available actions are bid 8, bid 6, and bid 4, our entries for S1 would become:

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

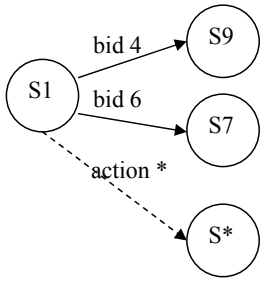

Figure 2: State action (crisp) pairs and Possible State action translations.

Furthermore, executing an action from S1, results in the agent moving into a new state. Sarsa is an on-policy algorithm. This means that the learning occurs only from experience. An on-policy learner selects an action, receives a reward and observes the new state. As with all reinforcement style algorithms, there must be a trade off between exploration and exploitation. An exploratory action and exploiting action is chosen as a result of the current policy. In the version of Sarsa implemented in our marketplace test bed, the action selection policy implemented was the ε greedy selection policy. ε greedy policy selection 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 (greedy selection). 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. In order to maximise exploration at the beginning of the simulation, an annealing factor is applied to ε until ε reaches a predefined minima (0.01). Thus the policy becomes:

γε	Randomly explore a different action.
(1-γε)	Make the greedy choice.

After an agent has executed an action in a particular state, the agent receives a reward. The rewards used in this simulation were based on overall achievement of the agents' goal. They can be summarized as follows:

$$\text{POSITIVE REWARD} = I_{won} * M_t / M_{t-a}$$
$$\text{NEGATIVE REWARD} = -I_{\underline{needed}}$$
$$\text{DEFAULT REWARD} = 0$$

Where I is the number items and M the amount of money the agent has. The agent receives a positive reward at the end of the episode if it has achieved its goal (i.e. it bought the number

of required items) and a negative reward if it has not. At all other non-terminating state-actions, the agent receives the default reward.

The algorithm then proceeds as follows:

```
All Q(s,a) values are initialised (to 0 in our case).
Repeat for each episode (or auction game){
      Initialize s_t (start state for the auction game).
      Choose a_t from s_t using ε greedy selection policy.
      Repeat for each step(auction) in the episode(auction game){
            Take action a_t, observe r and s_t+1
            Choose a_t+1 from s_t+1 using ε greedy selection policy
            Q(s_t, a_t) = Q(s_t, a_t) + α[r_t+1 + γQ(s_t+1, a_t+1) - Q(s_t, a_t)]
            s_t = s_t+1, a_t = a_t+1
      }
}
```

Figure 3: Sarsa Algorithm

4 Fuzzy Classifiers

In order to investigate fuzzy learning, the principles of fuzzification need to be investigated. As stated, 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 each of which a discrete number is mapped. The basic principle of fuzzification is to utilise fuzzy sets in state representation. For example, 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), $x \in \mathbb{Z}=[0..15]$. 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} and $\mu_{Items\_to\_Buy}$ might be described as:

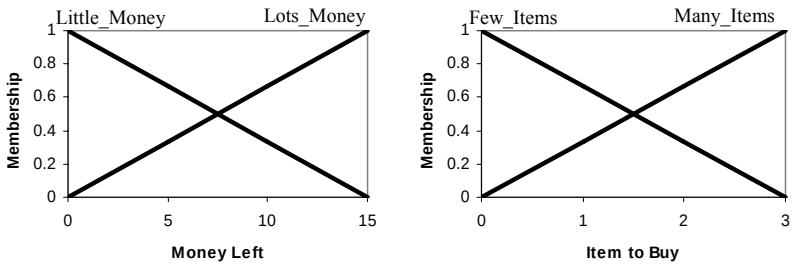

Figure 4: Membership function of Money_Left and Items_To_Buy

It is important to note that the membership functions can take on other patterns (bell, trapezoidal, etc) and can be additive (membership functions that total 1 for any given crisp value ($\Sigma_{i=1 \text{ to } n}\ \mu_i(x) = 1$)), or non-additive. However, it has been shown that systems that are additive are more robust to noise, design error etc. [7]. 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:

$$\mu_{Lots\_Money}(12) = 0.87 \text{ and } \mu_{Little\_Money}(12) = 0.13$$

In order to fuzzify a crisp state, the membership of each item of the state is fuzzified, and the AND of each item 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 5.

Fuzzy State	Money Left	$\mu_{\text{Money Left}}$	Items to Buy	$\mu_{\text{Items to buy}}$	μ_{S1}
$\hat{S}1_a$	Lots_Money	0.87	Few_Items	0	0
$\hat{S}1_b$	Lots_Money	0.87	Many_Items	1	0.87
$\hat{S}1_c$	Little_Money	0.13	Few_Items	0	0
$\hat{S}1_d$	Little_Money	0.13	Many_Items	1	0.13

Figure 5: Fuzzification of Crisp State S1

A system which uses a fuzzy representation of both states and actions will have entries along the lines of Figure 6. The membership calculations are still specific to the fuzzy state rather than state/action pairs, however membership functions for fuzzy actions are still required as they must be fuzzified and defuzzified before use.

Fuzzy State		Fuzzy Action
Money Left	Items to Buy	Bid
Lots_Money	Many_Items	Bid_High
Lots_Money	Many_Items	Bid_Low

Figure 6: Fuzzy state action pairs

For example, Bid_High might defuzzify, via the membership function, 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. As a fuzzy rule it is read as:

```
if Lots_Money and Many_Items then Bid_High
```

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 5 (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 (or the action proposed by the fuzzy rule with the highest QS-value). Therefore, the resulting action proposed, is a weighted average of the actions proposed by each rule triggered. These actions are weighted in terms of how appropriate each rule, or 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:

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

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.

	Fuzzy State		Fuzzy Action	
μ	Money Left	Items to Buy	Bid	FQ(ŝ,â)
0.7	Lots_Money	Many_Items	Bid_High	0.4
	Lots_Money	Many_Items	Bid_Low	0.1
0.4	Little_Money	Few_Items	Bid_High	0.2
	Little_Money	Few_Items	Bid_Low	0.6

Figure 7: Fuzzy state action pairs

Consider a crisp state s, which matches the two fuzzy states Lots_Money, Many_Items with degree 0.7 and Little_Money, Few_Items with degree 0.4. Each of these two fuzzy states have 2 rules associated with them. For the state Lots_Money, Many_Items, 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_Items, 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 7. The actual action taken is calculated as follows:

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

4.1 Fuzzy Q-Learning

Bonarini presents a different approach which attempts to account for more fuzzy principles than just fuzzy sets. In Bonarini's approach, crisp states are fuzzified into fuzzy states, and fuzzy states propose a fuzzy action. Both crisp states and actions are converted into fuzzy states and fuzzy actions. State fuzzification and action selection (following the centre of mass approach) occurs as discussed in Section 4. The fuzzy state actions or rules, are divided into *sub-populations* of rules. A sub-population is populated with rules that have the same antecedent (or state portion). For example, the rules in Figure 6 make up a sub-population. They all contain the same antecedent, but propose different consequents (or actions).

Bonarini presents several different forms of Fuzzy RL learning. In fuzzy versions of Q-learning and TD-λ are presented. The Q-learning algorithm, which we will base our extensions on, is presented here:

$$\hat{V}_k(\bar{r}) = \hat{V}_k(\bar{r}) + \alpha\xi_{r_j}(R_k + \gamma \max_i \hat{V}_{k+1}(r_i(s_{k+1}, a))\xi_i - \hat{V}_k(\bar{r})) \quad (2)$$

where $\hat{V}_k(\bar{r})$ is the value of rule r, R is the reward received, α is the learning rate, γ the future rewards discounting factor and $\xi_{c_{t-1}^i}$ the fuzzification factor. The primary new contribution from Q learning is the fuzzification factor and the update of all rules that are triggered (since in any given fuzzy state multiple rules are triggered to calculate the resulting action). The fuzzification factor weights the contribution of each fuzzy rule. The contribution is the degree of matching of the current state s to the fuzzy state ŝ ($\mu_{\hat{s}}{}^I(s_t)$). Divided by the sum of all degrees of matching for all fuzzy states ŝ that match the current state s ($\sum_{k=1,t}\mu_{\hat{s}^i}(s_k)$).

$$\xi_{(\hat{s}^i,\hat{a}^i)} = \frac{\mu_{\hat{s}^i}(s_t)}{\sum_{k=1,t}\mu_{\hat{s}^i}(s_k)} \quad (3)$$

5 Fuzzifying Sarsa

Section 3 discusses one type of algorithms that uses Q-learning algorithm for fuzzification. In the following sections we investigate two methods of performing Sarsa in our domain space. In Section 5.1 we investigate a fuzzy Sarsa algorithm, FQ Sarsa, that utilises fuzzy states. In 5.2 we present a fuzzy Sarsa algorithm, Fuzzy Sarsa, an on-policy fuzzy leaner which utilises fuzzy states and actions in the manner presented in 4.1.

5.1 FQ Sarsa

This algorithm is based on the Sarsa algorithm presented in 3. It is essentially Sarsa with the ability to cope with data stored as fuzzy sets (a fuzzy state), and therefore works by reducing the state space. It does not consider fuzzy actions or goal states, leaving these in their original crisp representation. In this approach, a crisp state `s` matches a set of fuzzy states and these fuzzy states are paired with crisp action values. Since the actions are not fuzzified, the selection mechanism operates on a simple greedy approach rather than using the fuzzy centre of mass approach. Therefore, 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 ($\max \mu(\hat{s})$, where μ is the degree of matching of crisp state *s* to fuzzy state $\hat{s}$). The FQ value update formula is modified as follows.

$$FQ(\hat{s}_{t-1}, a_{t-1}) = FQ(\hat{s}_{t-1}, a_{t-1}) + \alpha(r + \lambda FQ(\hat{s}_t, a_t)^{\wedge} \mu(\hat{s}_t) - FQ(\hat{s}_{t-1}, a_{t-1})) \quad (4)$$

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 then 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 are 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 selection 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 selection 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 8: FQ Sarsa Algorithm

5.2 Fuzzy Sarsa

The algorithm we present in Section 5.2 does not combine actions, it only selects them. This approach is problematic in that essentially the FQ Sarsa approach is only concentrating on reducing the state space and is not capable of leaning interactions between rules. To that effect, we now concentrate on the Fuzzy Q Learning algorithm presented by Bonarini and extend in order to implement it as an on-policy learner.

In this approach, states, actions and goals are all fuzzified in the manner described in 4.1. We use the centre of mass approach to calculate greedy actions. The Q value update formula is modified as follows:

$$FQ\ (\hat{s}^i_{t-1}, \hat{a}^i_{t-1}) = FQ(\hat{s}^i_{t-1}, \hat{a}^i_{t-1}) + \alpha\xi_{(\hat{s}^i_{t-1}, \hat{a}^i_{t-1})}(r_t + \gamma \sum_{\forall j} FQ(\hat{s}^j_t, \hat{a}^j_t)\xi_{(\hat{s}^j_t, \hat{a}^j_t)} - FQ_{t-1}(\hat{s}^i_{t-1}, \hat{a}^i_{t-1})) \quad (5)$$

where FQ values are the value of being in of fuzzy state and suggesting a fuzzy action, and $\xi_{c^i_{t-1}}$ is a fuzzification factor as described in Equation 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.

```
All Q(s,a) values are 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 9: Fuzzy Sarsa Algorithm

The algorithm presented in Figure 9 follows the same template as Sarsa. The primary differences between Sarsa and Fuzzy Sarsa lie in the state update as described above. After the learning algorithm is initialised and a start state found ($\hat{s}_t$), the algorithm chooses a fuzzy action ($\hat{a}_t$). This action is chosen based on the current policy, resulting in either an exploratory action or a greedy action. If an exploratory action is taken, the algorithm observes the result and updates *all matching* fuzzy state/action pairs ($\hat{s}_t$, $\hat{a}_t$), where $\hat{a}_t$ is the fuzzified crisp action taken, according to the FQ update equation in Equation 5. 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).

6 The Agent Marketplace

All three algorithms; Sarsa, FQ-Sarsa and Fuzzy Sarsa were implemented in an agent marketplace designed as discussed in Section 2. In the case of the Sarsa algorithm, we considered that the state of the world consisted 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. We elected to use four labels for each fuzzy category. Since membership functions are more robust when additive, $\Sigma_{i=1 \text{ to } n}\ \mu_i(x) = 1$, the functions we used were triangular (rather than trapezoidal, etc), since triangular membership functions are popular and easy to design additive functions with. Confirmation of the robustness of additive membership functions, came from the results of an earlier experiment using non-triangular and non-additive membership functions. During this test, our fuzzy algorithms were not able to find a solution, let alone an optimal one. The membership functions are given in Figure 10. FQ Sarsa does not use the Bid_Price function, since it utilises crisp bids.

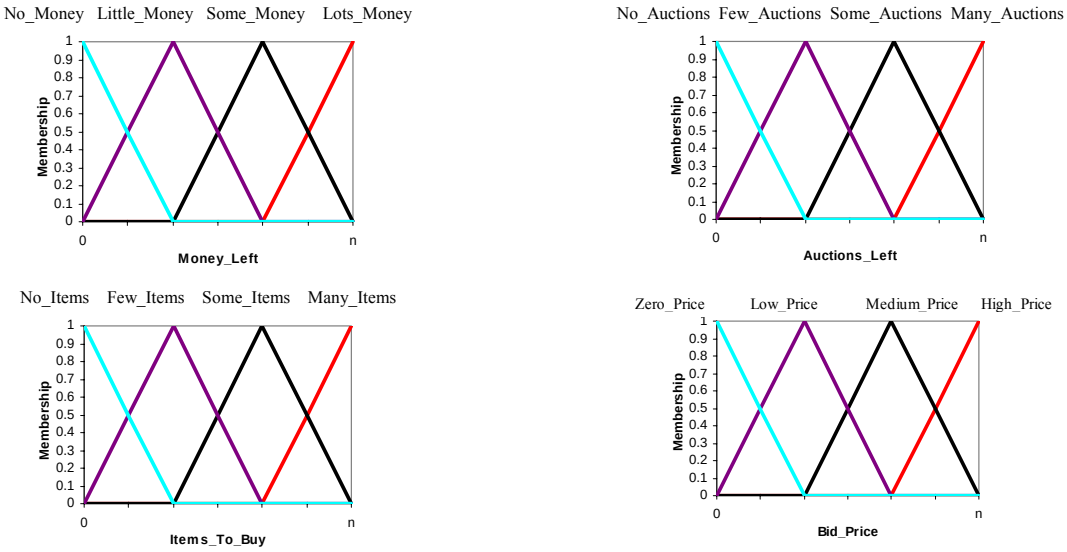

Figure 10: Fuzzy Membership Functions for the Test bed

In our experiment, all games were played with two agents; a fixed strategy agent, and a learning agent. The exploration and learning rates are both annealing parameters and gamma is set to 0.1. The results represent an average over 5 games. In test 1 of our algorithms, each agent must obtain 4 items over the 8 auctions in the episode. In this test, the price of each item ranges from 5 to 8.

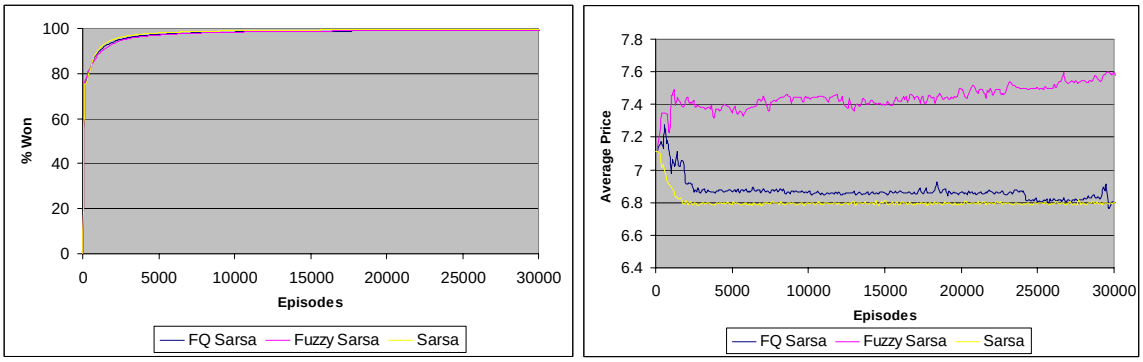

Figure 11: Percentage of Games Won in Test 1 and Quality of Solution in Test 1

As seen from Figure 11, all three algorithms converge upon a solution at similar rates. The difference in the algorithms can be seen from the quality of solution found. Both FQ Sarsa and Sarsa find solutions that are significantly better than that of Fuzzy Sarsa. We hypothesised that since fuzzy algorithms maximise learning around boundaries [7], that if the boundaries themselves do not represent a significant enough portion of the items, then the value of the solution may be affected. In test 1, the range of items to buy and the range of prices is equal to the range of fuzzy labels used, and therefore do not represent appropriate partitions in the fuzzy label boundaries. To determine if this was the case we performed a second test leaving all parameters constant except price, which was altered to have a larger range (5 to 12).

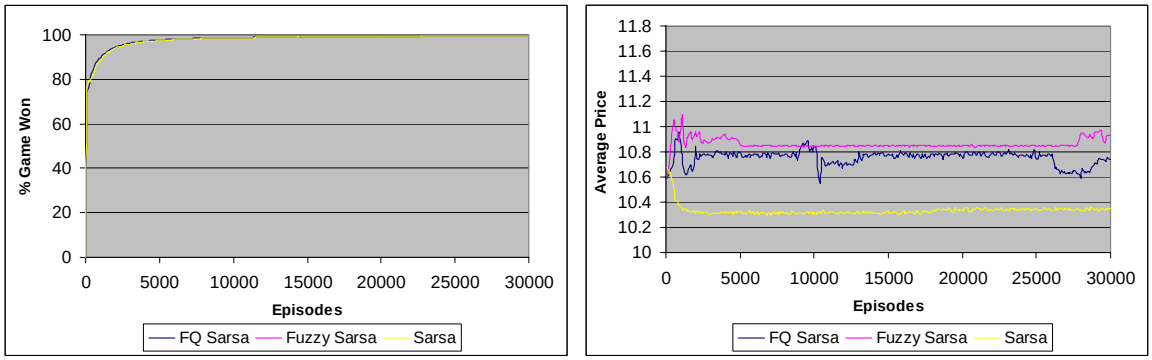

Figure 12: Percentage of Games Won in Test 2 and Quality of Solution in Test 2

As seen in Figure 12, the convergence rates remain similar to Test 1. However, as suspected the value of the solution found for Fuzzy Sarsa is more in line with what we would expect and closer to that of both FQ Sarsa and Sarsa. In order to confirm these results and that the overlap of the fuzzy boundaries on the fuzzy labels (ie. No_Items, Few_Items, etc) should be greater than the normal crisp labels (4 items to buy, 3 items to buy, etc), a further test was conducted. In this test, the number of auctions was increased to 12, the number of items required to 6 and the price range remained the same (5-12).

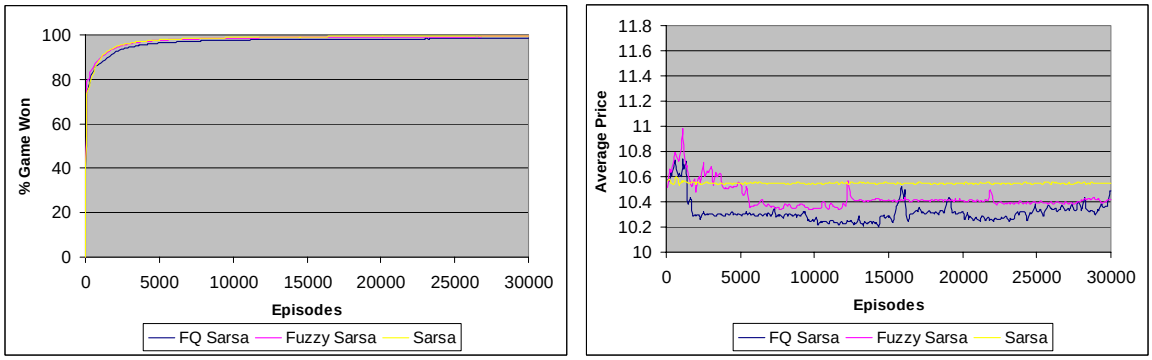

Figure 13: Percentage of Games Won in Test 3 and Quality of Solution in Test 3

In our final test, test 3, we were interested in the results of an even larger state space. For this test, we increase the number of items each agent must obtain to 10 items and the number of auctions to 20 (price remains the same at 5 to 12). As observed from Figure 14, both FQ Sarsa and Fuzzy Sarsa now converge to a solution quicker than Sarsa. As a result of the increased state space of Sarsa during this test, it has a tendency to get caught in a local minima if it does not stumble across a good solution during the exploratory stage of this algorithm. Furthermore, it is seen that the value of the solution found by Fuzzy Sarsa is superior to the one found by Sarsa, and that both Sarsa and Fuzzy Sarsa find superior solutions to that of FQ Sarsa. The reason for this is apparent, while FQ Sarsa converges quicker, it has a much reduced state

space than Sarsa, and thus does not have the range of possibilities available to Sarsa available to it. Again, we confirm that purer fuzzy solution presented by Fuzzy Sarsa, does seem to maximise transitions along fuzzy borders, allowing it to converge quicker and find a better solution than Sarsa.

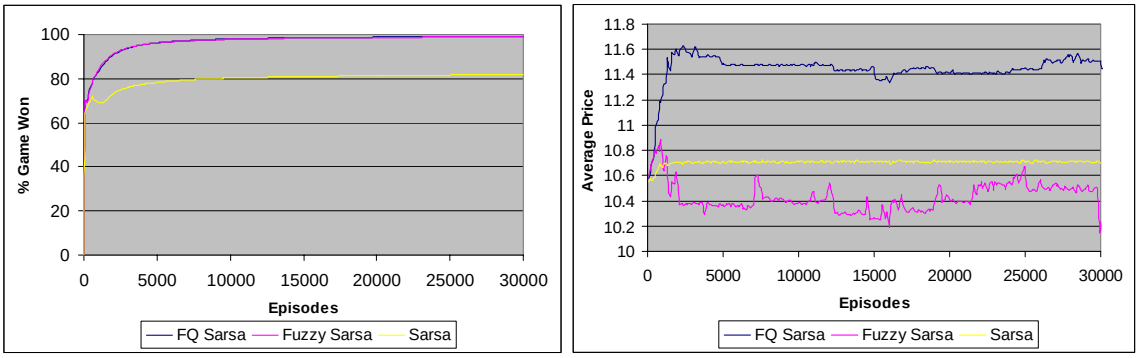

Figure 14: Percentage of Games Won in Test 4 Quality of Solution in Test 4

7 Conclusions

We have discussed some issues around applying fuzzy logic principles to both the reinforcement state action space and the algorithm itself. We found that Fuzzy Sarsa produces unpredictable and non-optimal results when the number of fuzzy labels used to encode information is greater than the number of actual labels. However, we have found that as the state space increases, a pure fuzzy logic approach to reinforcement learning as presented in Fuzzy Sarsa allows for a more robust and correct solution than the reduced state space algorithm presented by both Sarsa and FQ Sarsa. This is a significant result since Fuzzy Sarsa works with a significantly smaller state space than Sarsa. We plan to analyse Fuzzy Sarsa further, and to extend it to cope with coevolutionary learning.

References

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

[2] S. Noh and P. J. Gmytrasiewicz; *Agent Modelling in Antiair Defence;* Proceedings of the Sixth International Conference on User Modelling; 1997.

[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] 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.

[6] 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.

[7] 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.