FUZZY AND TILE CODING FUNCTION APPROXIMATION IN AGENT COEVOLUTION

ABSTRACT

Reinforcement learning (RL) is a machine learning technique for sequential decision making. This approach is well proven in many small-scale domains. The true potential of this technique cannot be fully realised until it can adequately deal with the large domain sizes that typically describe real world problems. RL with *function approximation* is one method of dealing with the domain size problem. This paper investigates two different function approximation approaches to RL: Fuzzy Sarsa and gradient-descent Sarsa(λ) with tile coding. It presents detailed experiments in two different simulation environments on the effectiveness of the two approaches. Initial experiments indicated that the tile coding approach had greater modelling capabilities in both testbed domains. However, experimentation in a coevolutionary scenario has indicated that Fuzzy Sarsa has greater flexibility.

KEY WORDS

Machine Learning, Fuzzy Logic, Reinforcement Learning, Function Approximation, Tile Coding, Coevolution.

1. Introduction

In order to move reinforcement learning (RL) out of "toy" domains, and realize its full potential in "real world" applications, RL must be able to efficiently cope with large state spaces. In the past few years there has been considerable research in the area of function approximation for RL. This is a particularly important area, as function approximation techniques are able to collapse the state space into a manageable size, and use their abilities to generalize between similar states and actions, to speed the learning process.

There has been significant work using linear function approximation, particularly in association with tile coding [1]. While a variety of different encoding techniques have been proposed, a good deal of success has been reported by the machine learning community with the method called tile coding [2][3][4]. This linear encoding technique collapses the state space down into a large feature vector and learns the associated weights of each feature. Another related solution, popular in the control system domain, is suggested by fuzzy set theory. A fuzzy set is a mapping from a set of real numbers to a set of symbolic labels. This approach stores states as fuzzy states. The basic principle of fuzzifying reinforcement learning is to utilise fuzzy sets in state representation. In this manner, many states can be represented with only a few fuzzy states. There have been several algorithms that utilise this idea and are presented as fuzzy reinforcement learning [5][6][7].

The primary contribution of this paper is the identification of a scenario in which the type of function approximation used makes a large difference. As shown, both methods perform reasonably well in stationary worlds, however in a coevolutionary context, Fuzzy Sarsa out performs tile coding.

The remainder of this paper is organized as follows. In Section 2, a review of the basic RL principles, fuzzy RL and tile coding is presented. Section 3 presents the two testbed domains. Section 4 presents the results of some experiments in these domains and Section 5, concludes with a summary and further areas for investigation.

2. Learning Algorithms

The following section begins with a brief overview of the principles of RL. Finally, it presents a synopsis of the two implemented function approximation algorithms, Fuzzy Sarsa and gradient-descent Sarsa(λ) with tile coding.

2.1 Reinforcement Learning

RL belongs to a family of *unsupervised learning* algorithms. Unsupervised algorithms force the learner to try to learn from its experiences. RL learners learn from simulated data. These algorithms build a mapping of situations to actions. A RL algorithm observes the current state of its world, and learns the best possible action from that state. The learner is never told what actions to take but rather what results are desired. It is up to the learner how they achieve the result. Reinforcement learners have 4 main elements: *a policy, a reward function, a value function and optionally, a model of the environment.* [1]. The typical objective for the learner is to develop a *policy* that maximizes long term return.

Basic RL is generally described as tabular, since its state/action space can be stored in table. Basic RL algorithms are limited when dealing with large, possibly continuous, state spaces. The size of lookup table that would be required, is not only computationally difficult, but results in very slow learning. A solution to this problem has been addressed by various different methods of function approximation.

2.2 Fuzzy Sarsa

Fuzzy Sarsa is based on the original Sarsa algorithm using fuzzification techniques presented in Bonarini's Fuzzy Q-Learning [8]. This algorithm is presented in full in [9]. This section will first review the principles of fuzzy logic applicable to Fuzzy Sarsa, before reviewing the Fuzzy Sarsa algorithm.

In fuzzy, reduction of the state space is based on storing the state/action space in fuzzy sets. Fuzzy sets are an extension of set theory that utilizes a set of fuzzy descriptors in place of the traditional set of crisp values. A fuzzy descriptor consists of a descriptor label to represent a range of crisp values. In the fuzzification process, a crisp state s matches a set of fuzzy states. 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 a world descriptor Money_Left. The value of Money_Left in a crisp state consists of a discrete number, say ML(x), $x \in Z = [0..15]$. 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. For example, the μ_{Money_Left} might be described as:

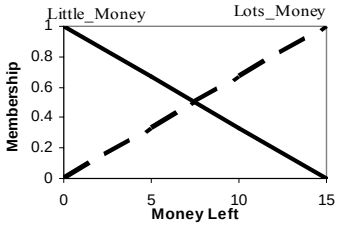

Figure 1: Membership function of Money_Left

The crisp values are fuzzified using these membership functions. Each crisp value will belong, to some degree, to one or more fuzzy set labels. The fuzzification of Money Left = 12 results in μ_{Lots_Money} (12) = 0.87 and μ_{Little_Money} (12) = 0.13. To fuzzify a crisp state, the membership of each state item is fuzzified, and, typically, the AND is taken to obtain the overall state membership or degree of matching. In the case of a crisp state S1={Money_Left = 12, Auctions_Left = 3}, 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 2.

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 2: Fuzzification of crisp state S1

Fuzzy Sarsa uses fuzzy representation of both states and actions. The degree of matching is still based on the fuzzy state, however membership functions for the fuzzification and defuzzification of fuzzy actions are also required. For example, Bid_High might defuzzify to the crisp action Bid 8. This fuzzy state action pair is referred to as a fuzzy rule. In a fuzzy rule, the fuzzy state corresponds to the antecedent of the rule and the fuzzy action is the consequent. All fuzzy rules have a strength associated with them. It is this strength, fuzzy Q (FQ) value that this type of fuzzy RL learns.

Greedy action selection for a fuzzy RL algorithm matches the crisp state *s* to one or more sets, sub-population, of fuzzy rules. Each sub-population is identified by having a common antecedent. The rule that is chosen from this sub-population, is the rule with the highest strength value. Since a crisp state *s* might match a number of fuzzy states (set $FS(s)$) as seen in Figure 2 (Both $\hat{S}1_b$ and $\hat{S}1_d$ match the fuzzy state S1), a method of combination is needed to determine what action to take when all rules could be proposing different actions. For all $\hat{s} \in FS(s)$, the action proposed for each $\hat{s}$, is the greedy action (highest FQ-value) proposed by the fuzzy rule. The final action proposed is a weighted average of the actions, in terms the degree of matching to the crisp state s, proposed by each rule triggered. The weighted average is computed using the centre of mass approach:

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

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 $\hat{s}_i$.

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 3: Fuzzy state action pairs

To clarify greedy action selection, consider the example from Figure 3. Some crisp state s, 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 via some defuzzification function as bid 8 and Bid_Low as bid 4. Thus the actual action taken is calculated as follows would be 6.5. For Fuzzy Sarsa, the FQ value update formula is as follows:

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

where $FQ\ (\hat{s}^i_{t-1}, \hat{a}^i_{t-1})$ is the value of being in the fuzzy state $\hat{s}^i_{t-1}$ and suggesting a fuzzy action $\hat{a}^i_{t-1}$. $\xi_{\hat{s}^i_{t-1}}$ 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:

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

The full Fuzzy Sarsa algorithm is presented in [9]

2.3 Gradient-descent Sarsa(λ) and tile coding

Gradient-descent methods rely on the state space being encoded in a parameter vector of features, $\vec{\theta}$. Any state can be described by one or more of these features. In gradient-descent methods, since the state space is represented by a parameter vector $\vec{\theta}$, the Q(s,a) value is calculated by using the sum of $\vec{\theta}$ values present for that state/action pair. In the case of the marketplace example, the state action pair S1={Money_Left = 12, Auctions_Left = 3}, a = {6} now has its own vector of features $\vec{\phi}_s$. This vector is a big binary vector that indicates whether the corresponding feature in $\vec{\theta}_s$ is present in the state. The Q(s,a) value is calculated as:

$$Q(s,a) = \sum_{j=0}^{s} \theta_j \vec{\phi}_j$$

Most gradient-descent methods try to minimise the error on the observed examples by adjusting the entry for the present feature by a small amount. While there are both on-policy and off-policy gradient-descent methods, this study uses on-policy algorithms. Gradient-descent Sarsa(λ) is given in full in [1].

One of the most important aspects of gradient-descent methods is feature selection. A variety of coding techniques for feature selection such as radial basis functions, kanerva coding, etc, have been introduced. A good review of these techniques is proved by [1]. In particular, many researchers have reported success with tile coding [2][3][4] and therefore it has an established set of published results for comparison.

Tile coding, or CMAC, was first introduced by Albus [10]. Over the last few years it has been adapted for use in reinforcement learning and renamed tile coding [1]. The basic principle behind tile coding is to overlay the state spaces with exhaustive partitions. Each partition is called a *tiling*, and every element in the partition a *tile*. Each tile makes up one feature and the total set of tiles in all tilings $\vec{\theta}$. The resolution is divided into generalisation and granularity parameters. The generalisation parameter describes the shape of the tiles. The granularity parameter is described by the number of tiling overlaying the state space. These overlays are important in tile coding's ability to make fine distinctions. The combination of generalisation and granularity is called the overall resolution. If a state space is described by two state variables x and y, one possible way to tile it is to create 4x4 regions across the state space. This creates broad generalisation between state values that are within 0.25 of each other (in both x and y). This level of generalisation is relatively coarse. To refine the detail of what is learnt, another tiling offset from the original can be placed over the state space. Figure 4 shows the original 2-dimensional state space with 2 offset 4x4 tilings. The example state lies in exactly one tile in each tiling. Generalisation of that state occurs with any other state that lies within that tile. Since the offset is different for each tiling, the cluster of states surrounding the original state differs.

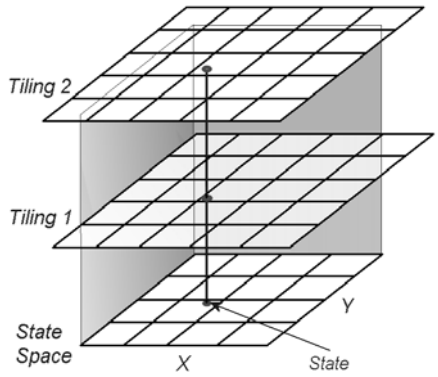

Figure 4: Determining the tiles of a state

The overall resolution of this example is 0.25/2 or 0.125. Finer resolution can be achieved by increasing the number of tilings. In summary, the shape and size of the tiles determines the type of generalisation that occurs between states, whereas the number of tiling overlays controls the distinctions made about them.

3. Test Domains

This section presents the two testbed domains used in the experiments.

3.1 Mountain-Car World

The mountain-car problem is described in detail by [1]. In this problem the learner controls an underpowered car that is situated in a deep valley. The objective is to get the car to the top of the mountain. The difficulty is that the car is underpowered and thus cannot gain enough momentum by simply going forward to get it to the top of the mountain. In order to find a solution, the learner must first move away from the goal. This type of problem is one that RL typically find difficult. The actions in this world are: full throttle forward, full throttle backwards, or zero throttle. The car moves according to:

$$p_{t+1} = bound[p_t + p_{t+1}]$$
$$v_{t+1} = bound[v_t + 0.001a_t + -0.0025\cos(3v_t)]$$

Where p_t and v_t is the car's position and velocity at time t and a_t, the action taken. The bound operation

enforces $-1.2 \le p_{t+1} \le 0.6$ and $-0.07 \le v_{t+1} \le 0.07$. The rewards in this domain are -1 at all non-terminal states. This environment serves as the tile coding control world, as the implementation is based on the published code of the mountain-car world including the gradient-descent Sarsa(λ) with tile coding learner, provided by Sutton [12].

3.2 An Agent Marketplace Testbed

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, termed 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 term *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 rewards at the terminal states are $I_{won} * M_t / M_{t-a}$ if the agent has purchased all items required and $-I_{needed}$ if not. At all non-terminating state-actions, the agent receives a default reward of 0.

This domain was chosen for two reasons. The first reason is the added complexity of the state/action and goal space. Although variables in the marketplace domain are not continuous, the state/action space is considerably different from the mountain-car world. Auctions have three different state variables, rather than the two of the mountain-car world. This increase tests the modeling abilities of each type of function approximation method. The action space, price to bid, is a more populated action space than that of the mountain-car world (forward, reverse, no throttle). Finally, in the marketplace world, there are many possible goal states, whereas the mountain-car world only has one. The second reason for choosing the marketplace domain is that it is a domain in which Fuzzy Sarsa has performed reasonably. [9]

4. Experimental Results

4.1 Experiments in the Mountain-car World

For the first experiment to determine the effectiveness of the two different forms of function approximation RL, the Fuzzy Sarsa algorithm was implemented in the mountain-car world. The fuzzy algorithm was then compared to the established tile coding example. The gradient-descent Sarsa(λ) with tile coding used 9x9 grid tilings, as described in [1]. As depicted in Figure 5, the fuzzy algorithm used 7 label velocity and position state variables, and 2 label actions.

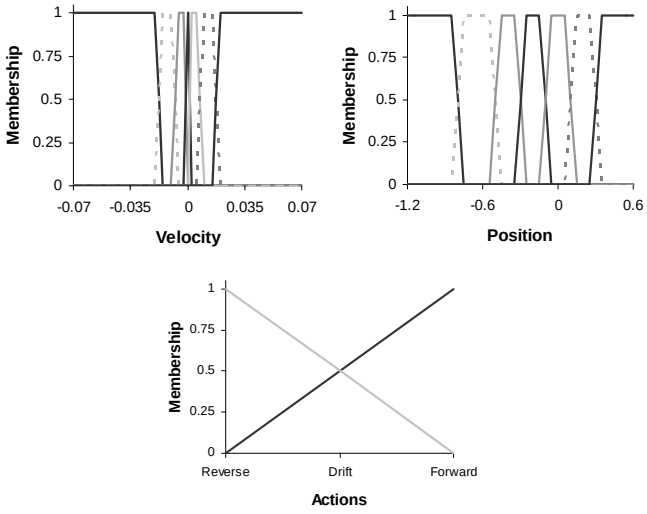

Figure 5: Membership functions for mountain-car world.

Using the parameters γ=1, α=0.6 and ε = 0.025 (with annealing for epsilon to allow the algorithms to settle on a greedy policy), both algorithms found a solution. Figure 6 gives the average solution found over 101 episodes. While both algorithms perform well in this domain, gradient-descent Sarsa(λ) with tile coding achieves a marginally more optimal and stable solution.

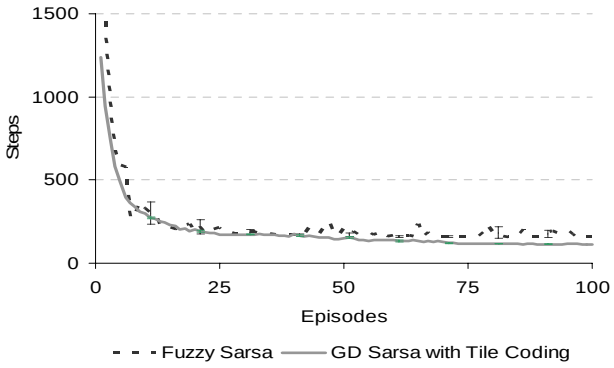

Figure 6: Stability of solution in Mountain-car problem

To investigate the reason behind this improvement, the final action policy for each algorithm is illustrated by the two surface maps in Figure 7.

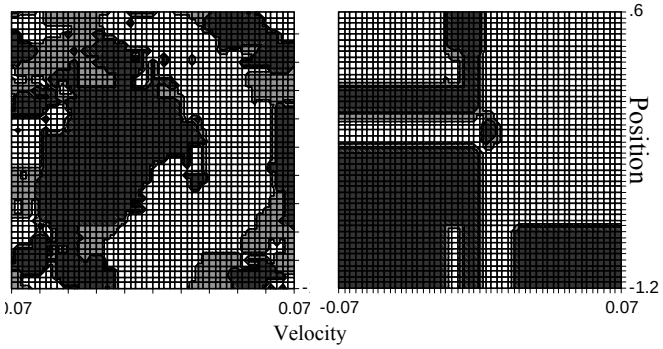

Figure 7: Final action policies for mountain-car world.

The surface maps indicate that both algorithms have learnt policies that have broad areas of similarity. However, it is apparent that the policy learnt by Fuzzy Sarsa lacks the fine distinctions apparent in the policy of tile coding.

4.2 Experiments in the Marketplace

The state space consists of 3 major categories: Money_Left, Auctions_Left and Items_Left. Actions include bids ranging from the original offer price to the agent's maximum price and abstaining. Each category is divided into three labels. The general membership function used for all state and action variables is given in Figure 8.

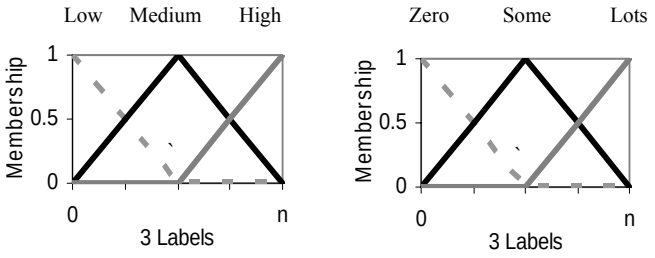

Figure 8: Fuzzy membership functions for the marketplace

The tile coding setting used was 2 tilings of 2x2x2. The RL parameters were $\alpha = 0.2$, $\varepsilon=0.03$, and $\gamma = 1.0$. The settings were determined following a "good enough" methodology. For the fuzzy memberships, several different combinations were trialed before deciding on the three label functions. For the tile coding settings, a variety of candidate settings were empirically evaluated in the marketplace before choosing the most effective setting. The RL parameter settings were chosen by using the top ten best parameters found for both algorithms using a scanning method. From both sets of top ten parameter settings, the top parameter in common was selected.

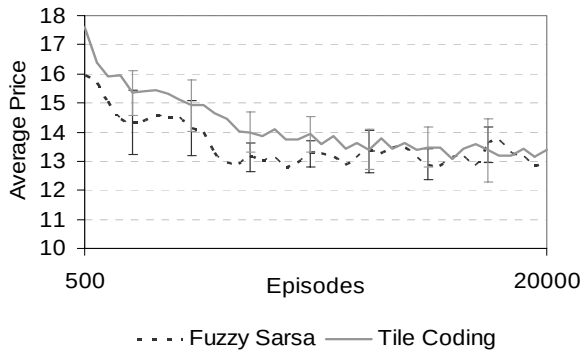

Figure 9: Learners against a linear strategy

For the experiments, an auction game consisted of 40 auctions, of which, each agent was required to win 20 items. Each function approximation learner is tested against two stationary strategies, Linear and Greedy. An agent following a stationary strategy, follows the same behavioural policy in every auction game. A linear strategy slowly increases its bid over time, while a greedy strategy bids the maximum it can until it has obtained all needed items. The results presented are averaged over 10 trials. Figure 9 indicates that both algorithms perform similarly when competing with a linear strategy, whereas Figure 10 indicates that the solution found by tile coding offers a small, but significant improvement over that of the fuzzy approach.

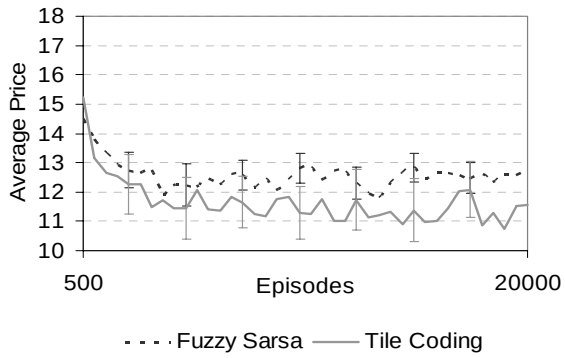

Figure 10: Learners against a greedy strategy

4.3 Coevolution experiments in the Marketplace

The results in both the mountain-car and the marketplace agree with the recent findings of [11], in their analysis of the abilities of fuzzy and tile coding function approximation in the learning of a selection of functions. This research investigates one type of test that could not be done in a simple function learning problem, or in the mountain-car world. This test consists of examining the abilities of both methods in a coevolutionary context.

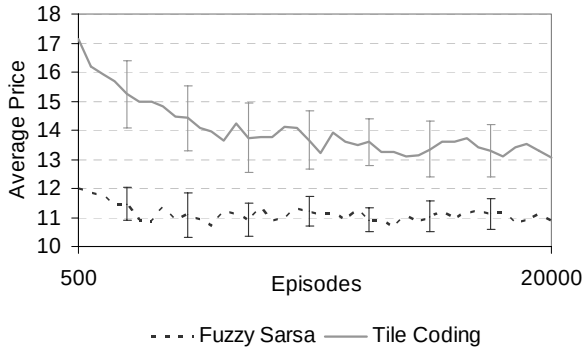

Figure 11: Fuzzy Sarsa vs. Tile Coding in the Marketplace

Figure 11 presents the results of Fuzzy Sarsa vs. gradient-descent Sarsa(λ) with tile coding in the same type of marketplace as the previous tests. In this experiment, the two algorithms are the only two agents competing in the market, and therefore, must learn in tandem. There are enough items for both agents, and thus the goal of the two algorithms is to learn how to divide the items between them. Fuzzy Sarsa clearly achieves a significantly better price than tile coding throughout the experiment. Given the tile coding agent's better performance in fixed strategy experiments, this poor performance is somewhat unexpected. Before concluding that Fuzzy Sarsa has

more powerful modelling capabilities in a coevolutionary context, the poorer performance of the tile coding agent must be investigated.

In the previous experiment, tile coding vs. a stationary strategy, there is only one learner. Therefore, the interactions of the game remain constant and the tile coding agent is able to refine its fairly course state space representation to learn a good strategy to use in a stationary context. In the coevolution experiment, because more than one agent is learning at the same time, the interactions of the game fluctuate. This fluctuation makes it more difficult for each agent to learn an optimal solution. As indicated in Figure 11, the prices achieved by the tile coding agent fluctuate more than those achieved by the Fuzzy Sarsa agent. One possible reason for the poorer performance of the tile coding agent is that the generalisation and resolution setting chosen by scanning a variety of combinations in a stationary strategy environment does not allow enough resolution to provide for the types of distinctions needed to perform well in the coevolution experiment. To determine if the tile coding settings were affecting the agent's results, the experiment was rerun with a selection of other candidate settings.

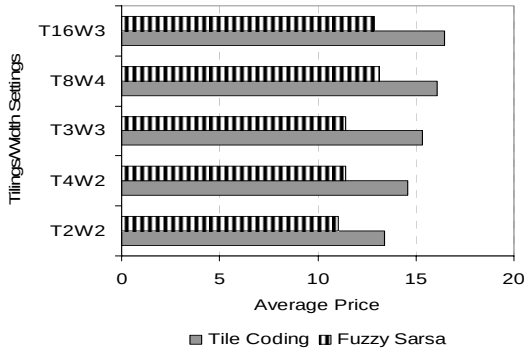

Figure 12: Other tiling settings in the marketplace coevolution test

Figure 12 presents the averaged price achieved in the last 5000 episodes (out of 20000) by the tile coding agent and the Fuzzy Sarsa agent when in direct competition with each other. For both agents, the optimal price achieved is when the Tile Coding agent is at T2W2. In this environment, stiff competition increases the performance of Fuzzy Sarsa. It pushes the algorithm to carefully refine its distinctions between both cooperative state/action pairs, pairs that work together to generate a solution, and competitive ones, pairs that contain the same state portion.

Since the speed of learning in each individual tile in the tile coding method is dependant on the number of tilings, in a one further attempt to boost the algorithm's performance the T8W4 test was rerun with increased α setting. However, with α= 0.8, the performance of the tile coding agent did not improve.

5. Conclusion

This paper has demonstrated that the performance of a function approximation method is dependant on the type of environment. Through a detailed, both fuzzy and tile coding techniques were shown to perform similarly in most tests. However, the fuzzy approach was shown to be more robust with regards to coevolution. While further parameter tuning may boost the performance of the tile coding algorithm in the coevolutionary experiment, in is worth noting that this is not required of the fuzzy technique. This makes the fuzzy technique easier to use. Further investigations into this problem will examine potential tuning issues and investigate the function approximation methods in other types of coevolutionary scenarios such as different types of competitive coevolution and cooperative coevolution.

References

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

[2] R. S. Sutton; Generalization in reinforcement learning: Successful examples using sparse coarse coding; *In Advances in Neural Information Processing Systems, volume 8;* The MIT Press, 1996.

[3] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement Learning for RoboCup-Soccer Keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[4] Alexander A. Sherstov and Peter Stone. Function Approximation via Tile Coding: Automating Parameter Choice. *In SARA 2005,* pp. 194–205, Springer Verlag, Berlin, 2005.

[5] P. Y. Glorennec. Fuzzy Q-learning and Dynamical Fuzzy Q-Learning. *In FUZZ-IEEE*, Orlando, FL. 1994

[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] 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

[10] Albus, J. S; *Brains, Behavior, and Robotics. BYTE*/McGraw-Hill, Peterborough, 1981.

[11] Lashon B. Booker, *Approximating Value Functions in Classifier Systems,* technical paper, The MITRE Corporation, February 2005

[12] Mountain-car C++ and Lisp Implementation http://www.cs.ualberta.ca/~sutton/MountainCar/MountainCar.html