FUZZY STATE AGGREGATION AND OFF-POLICY REINFORCEMENT LEARNING FOR STOCHASTIC ENVIRONMENTS

Dean C. Wardell and Gilbert L. Peterson
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH, 45433
USA
dean.wardell, gilbert.peterson@afit.edu

ABSTRACT
Reinforcement learning is one of the more attractive machine learning technologies, due to its unsupervised learning structure and ability to continually learn even as the environment it is operating in changes. This ability to learn in an unsupervised manner in a changing environment is applicable in complex domains through the use of function approximation of the domain's policy. The function approximation presented here is that of fuzzy state aggregation. This article presents the use of fuzzy state aggregation with the current policy hill climbing methods of Win or Lose Fast (WoLF) and policy-dynamics based WoLF (PD-WoLF), exceeding the learning rate and performance of the combined fuzzy state aggregation and Q-learning reinforcement learning. Results of testing using the TileWorld domain demonstrate the policy hill climbing performs better than the existing Q-learning implementations.

KEY WORDS
Reinforcement Learning, Policy Hill-Climbing, Fuzzy State Aggregation, Stochastic Environment

1. Introduction

As researchers in the field of machine learning tackle more and more complex problems, the obstacle of ever increasing state-space sizes is a constant challenge. Simply improving the speed of the algorithms frequently cannot overcome the enormity of the state-space and provide useful results in a timely manner.

Using a state generalization architecture to limit the size of the state space and approximate the learned policy has been presented in several previous efforts [1, 2, 3]. Specifically, Berenji and Vengerov [4, 5] use fuzzy state aggregation (FSA) as a means of effectively limiting the state space in a Q-learning experiment.

One method of improving the results of Q-learning consists of adding the use of a separate policy table to track the probability of selecting an action from a given state. The off-policy reinforcement learning algorithm, policy hill climbing, yields improved empirical results over on-policy methods [1]. Bowling and Veloso [6] showed continued improvement over policy hill climbing (PHC) by separating the delta update value into two values, one which updates when winning and one for loosing hence the name of Win or Lose Fast (WoLF) policy hill climbing algorithm. Banjeree and Peng extend WoLF, introducing a policy dynamics based version of WoLF (PDWoLF), further improving results [7].

In this paper we present an application of fuzzy state aggregation combined with three different policy hill-climbing algorithms comparing the speed and efficacy of their learning in the highly stochastic TileWorld [8] environment. The results demonstrate the improved performance of combining an off-policy reinforcement learning method with FSA.

In section 2 of this paper we provide an overview of these different methods and algorithms. Section 3 covers the combinations of the FSA vector with PHC learning. The TileWorld domain is described in section 4 and the experiments conducted and results follow.

2. Background and Related Work

State aggregation is a type of generalizing function approximation which allows machine learning to learn in larger environments more quickly. State aggregation works by combining the states of a domain into groups with some common value estimate [1]. When a state is updated, the entire group is updated. The best known methods for state aggregation are tile coding (also known as sparse coarse coding) [1,9] artificial neural networks [2], and fuzzy state aggregation [4,5]. In the following section, fuzzy state aggregation is described.

2.1 Fuzzy State Aggregation

Fuzzy state aggregation uses Zadeh's [10] concept of fuzzy sets to represent the environment with a limited number of "fuzzy states". Fuzzy sets are sets that allow elements to be partially in more than one set at a time. The degree to which an element is a member of a fuzzy set is measured on a scale between 0 and 1.

For example, consider the outside ambient temperature [11]. Classical set theory can only classify the temperature as hot or cold (*i.e.*, either 1 or 0). It cannot interpret the temperature between 20°F and 100°F.

In other words, the characteristic function for the classical logic for the above example is given by

$$\mu_{Hot}(x) = \begin{cases} 1 \ iff \ \ x \geq 50^{o}F \quad classified \quad as \ \ hot \\ 0 \ iff \ \ x < 50^{o}F \quad classified \quad as \ \ cold \end{cases}$$

The boundary 50 °F is taken because classical logic cannot interpret intermediate values. On the other hand, fuzzy logic solves the above problem with a membership function as given by

$$\mu_{Hot}(x) = \begin{cases} 0 & if \ \ x < 20^{o}F \\ \dfrac{x-20}{80} & if \ \ 20^{o}F \leq x \leq 100^{o}F \\ 1 & if \ \ x > 100^{o}F \end{cases}$$

The above membership function is graphed in Figure 1. Demonstrating that the degree of coldness is the complement of the degree of hotness.

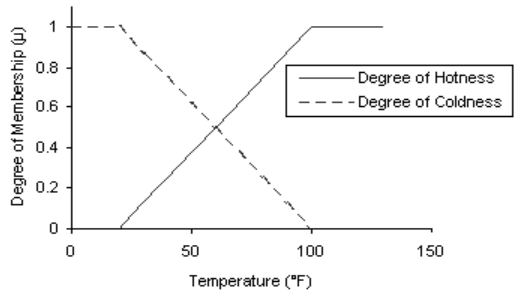

Figure 1: Membership function for the degree of hotness and degree of coldness [11]

Fuzzy state aggregation is a variation of Singh's soft state aggregation [3], which uses probability values as a measure of the extent to which the current state falls into the various aggregate (cluster) states.

Like soft state aggregation, fuzzy state aggregation uses a fixed number (*K*) of aggregate states to represent the environment and thus minimize the number states the learning algorithm must deal with. Rather than using probabilities, a crisp state (*s*) is represented by its degree of simultaneous membership in each of the *K* fuzzy states. The total number of fuzzy states is determined by the number of fuzzy sets (labels) used and the number of state variables.

2.3 Q-Learning

In the realm of reinforcement learning, Q-learning [12] is one of the simplest and most commonly used methods. Q-learning assigns values to state-action pairs Q(s,a), and thus implicitly represents a policy. After the algorithm selects an action, *a*, the Q table representing the policy is updated based on the rewards received and the expected rewards as represented by the Q value of the resultant state, *s'*, according to the function:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \quad (1)$$

where α is the learning rate (or step size), between 0 and 1, that controls convergence, and γ is the discount factor, between 0 and 1, that makes rewards *r* that are earned later exponentially less valuable

In Q-learning, the agent learns through continuous interaction with the environment, during which it exploits what it has learned so far. To ensure the agent is not missing more valuable state-action pairs, it can also explore. In practice, this means that the current approximation *Q* is used to select an action most of the time. However, in a small fraction of cases an action is selected randomly from the available choices, so as to explore and evaluate unseen state/action pairs.

2.4 Combining state aggregation with Q-learning

In a domain with a large state-space, it is very inefficient to learn separate Q-values for each state-action pair. It is therefore, not uncommon to see Q-learning used in conjunction with some form of state aggregation. When implementing Q-learning with such an architecture, the term *Q(s,a,r)* is used to approximate *Q(s,a.)* Here *r* is a vector of the learned parameters. The fundamental parameter updating rule for each time step *t* is [4]:

$$r_t \leftarrow r_t + \alpha\delta_t \Delta r_t Q(s_t, a, r_t) \quad (2)$$

Where α is the learning rate and δ_t is the Bellman error used for the look-up vector in this corresponding learning rule:

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha\delta_t \quad (3)$$

In discounted Q-learning the Bellman error is calculated as follows:

$$\delta_t = g(t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a) \quad (4)$$

Where *g(t)* is the cost of taking the specific action and γ is the discount rate.

In this work we have specifically used fuzzy state aggregation as the function approximation architecture. Using this architecture, the Q-value of action *a* in state *s* is calculated using:

$$Q(s,a) = \sum_{k=1}^{K} q(k)\mu_k(s,a) \quad (5)$$

Where *q(k)* is the Q-value of the k^{th} fuzzy state and $\mu_k(s,a)$ is the degree of membership of state *s* to *k* with respect to action *a*.

Replacing $\Delta r_t Q(s_t,a,r_t)$ from equation (2) with $\mu_k(s,a)$, the equation to update *q(k)* becomes:

$$\forall_{k \in K} \quad q(k) \leftarrow q(k) + \alpha\delta_t \mu_k(s,a) \quad (6)$$

Otherwise, the Q-learning algorithm remains unchanged.

2.5 Policy Hill Climbing

Policy Hill Climbing (PHC) is a simple extension of Q-learning. The algorithm, performs hill-climbing (seeking the highest global reward) in the space of mixed policies. Q-values are maintained as an estimate of the optimal policy. In addition to the Q-table, the algorithm

maintains the current mixed policy (policy table). The PHC algorithm is shown in Table 1.

Input $\alpha \in (0,1]$, $\delta \in (0,1]$.

Initialize $Q(s,a) \leftarrow 0$ *and* $\pi(s,a) \leftarrow \frac{1}{|A|}$.

Loop

from state s, select action a according to $\pi(s,a)$

Observe r and next state s' and update $Q(s,a)$

compute

$$\Delta_{sa} = \begin{cases} -\delta_{sa} & \text{if } a \neq \max_a Q(s,a') \\ \sum_{a' \neq a \in A} \delta_{sa} & \text{otherwise} \end{cases}$$

$$\text{where } \delta_{sa} = \min\left(\pi(s,a), \frac{\delta}{|A|-1}\right)$$

Update $\pi(s,a) \leftarrow \pi(s,a) + \Delta_{sa}$

Table 1: Basic Policy Hill Climbing

The policy is improved by increasing the probability that it selects the highest valued action according to a learning rate $\delta \in (0,1]$. When $\delta = 1$ the algorithm is equivalent to Q-learning, since with each step the policy moves to the greedy policy, always executing the highest valued next step rather than pursuing the greatest overall reward.

2.6 Win or Lose Fast (WoLF-PHC)

The WoLF-PHC [6] algorithm is a hill climber that also uses a variable learning rate. The algorithm requires two learning parameters $\delta_l > \delta_w$. The parameter that is used to update the policy depends on whether the agent thinks it is currently winning or losing. This determination consists of comparing whether the current expected value is greater than the current expected value of the average policy. If the current expected value is lower (i.e., the agent is "losing"), then the larger learning rate δ_l is used, otherwise δ_w is used. The purpose of using the variable learning rate is to increase the speed at which the algorithm reaches the optimum policy. The functions below are used to calculate δ for the WoLF-PHC algorithm, and are the only changes to PHC in Table 1.

$$\bar{\pi}(s,a) \leftarrow \bar{\pi}(s,a) + \left(\frac{\pi(s,a) - \bar{\pi}(s,a)}{C(s)}\right)$$

$$\delta = \begin{cases} \delta_w & \text{if } \sum_a^A \pi(s,a)Q(s,a) > \sum_a^A \bar{\pi}(s,a)Q(s,a) \\ \delta_l & \text{otherwise} \end{cases}$$

Table 2: Additional functions for WoLF-PHC

2.7 Policy Dynamics based Win or Lose Fast (PDWoLF)

Like WoLF, PDWoLF [7] uses the variable learning rate parameters δ_l and δ_w. Where WoLF checks itself against an average policy to determine if it is winning or losing, PDWoLF uses the change in policy from the previous time step $\Delta(s,a)$ with the change in policy from the current time step Δ_{sa}, shown below.

$$\delta = \begin{cases} \delta_w & \text{if } \Delta(s,a)\,\Delta^2(s,a) < 0 \\ \delta_l & \text{otherwise} \end{cases}$$

$$\text{where } \Delta^2 \leftarrow \Delta_{sa} - \Delta(s,a), \text{ and } \Delta(s,a) \leftarrow \Delta_{sa}$$

Table 3: Additional functions used in PDWoLF-PHC

3. Combining PHC and FSA

The use of state aggregation for function approximation with Q-learning is not a new or unusual concept [1, 3]. Berenji and Vengerov [4, 5] advanced this work in their application of Q-learning and fuzzy state aggregation. We have built upon their work by beginning with fuzzy state aggregation and a basic Q-learning algorithm and extending that to the application of PHC algorithms. With the state-space constrained to K total fuzzy states, we applied three different variants of a Policy Hill Climbing algorithm; standard PHC, Win or Lose Fast (WoLF) PHC and Policy Dynamics (PD) WoLF-PHC. Our implementation of these algorithms uses two vectors to represent the learned parameter data. The q-vector $q(k)$ as described previously and a policy vector $\pi(k)$.

The q-vector holds the expected reward over time which is iteratively updated using a common temporal-difference formula. The π-vector holds the probabilities used to select an action from a given state (the policy). The policy decision of which action to take next is then based on both the expected reward value (q) and the policy value (π):

$$\Pi(s,a) = \sum_{k=1}^{K} \pi(k) q(k) \mu_k(s,a) \tag{7}$$

The vectors $q(k)$ and $\pi(k)$ are initialized as shown below:

$$q(k) \leftarrow 20 \text{ and } \pi(k) \leftarrow \frac{no.\ of\ fuzzy\ labels}{|AK|} \tag{8}$$

where A is the number of possible actions in the domain. The reason for initializing $\pi(k)$ this way may not be intuitively obvious. Since we are using 3 fuzzy labels, the initial value of each element of $\mu_k(s,a)$ is 1/3 before learning begins. The elements of $\pi(k)$ are initialized so that

$$1 = \sum_{a=1}^{A} \sum_{k=1}^{K} \pi(k) \mu_k(s,a) \tag{9}$$

Normalizing it with a Boltzmann distribution to ensure equation (9) remains true, the π-vector is updated as follows:

$$\forall_{k \in K} \quad \pi(k) \leftarrow \pi(k) + \left[\frac{\Delta_{sa}}{K} \mu_k(s,a)\right] \tag{10}$$

with Δ_{sa} calculated as:

$$\Delta_{sa} = \begin{cases} -\delta_{sa} & if\ a \neq \max_a Q(s,a') \\ \sum_{a' \neq a \in A} \delta_{sa} & otherwise \end{cases} \quad (11)$$

Where

$$\delta_{sa} = \min\left(\sum_{k=1}^{K} \pi(k)\mu_k(s,a), \frac{\delta}{|A|-1} \right) \quad (12)$$

In this application δ is set in the range (0,1].

Because our intent is to use Δ_{sa} to update the entire summation

$$\sum_{k=1}^{K} \pi(k)\mu_k(s,a) = \sum_{k=1}^{K} \pi(k)\mu_k(s,a) + \Delta_{sa} \quad (13)$$

for a given action *a* the division used in equation (10) is necessary to scale and weight Δ_{sa} correctly and prevent it from causing disproportionate growth in the elements of $\pi(k)$.

3.1 Combining WoLF-PHC and Fuzzy State Aggregation

Unlike the standard PHC algorithm, the WoLF-PHC and PDWoLF-PHC both utilize a dynamic learning rate to increase the speed of convergence over the standard PHC.

The WoLF-PHC algorithm uses an additional vector to estimate the average policy value. The average policy vector is initialized like the π-vector:

$$\bar{\pi}(k) \leftarrow \frac{no.\ of\ \ fuzzy\ labels}{|AK|} \quad (14)$$

This vector is updated by

$$\forall_{k \in K} \quad \bar{\pi}(k) \leftarrow \bar{\pi}(k) + \left(\frac{\pi(k) - \bar{\pi}(k)}{C} \right) \quad (15)$$

where *C* is a counting function used to track how many times the elements representing a state have been updated. In this implementation all state elements for the selected action are updated simultaneously, so *C* is simply the number of times the algorithm has looped.

The delta selection for determining the learning rate in WoLF is then calculated as follows:

$$\delta = \begin{cases} \delta_w & if \sum_{k=1}^{K}\sum_{a=1}^{A} \pi(k)q(k)\mu_k(s,a) > \sum_{k=1}^{K}\sum_{a=1}^{A} \bar{\pi}(k)q(k)\mu_k(s,a) \\ \delta_l & otherwise \end{cases} \quad (16)$$

where $\delta_l > \delta_w$ and both fall within the range (0,1]. This value for δ is used to calculate Δ_{sa} as described in equation (10) and is derived from the δ calculation of WoLF in equation 16.

3.2 Applying PDWoLF to the FSA

The PDWoLF-PHC also uses additional values to change the learning rate. These are initialized as

$$\Delta(s,a) \leftarrow 0 \ \text{ and } \ \Delta^2(s,a) \leftarrow 0 \quad (17)$$

Where Δ and Δ^2 are changing rates within the policy and are estimates of the slopes of the decision space. These are respectively updated for the selected action as

$$\begin{aligned} \Delta^2(s,a) &\leftarrow \left(\sum_{k=1}^{K} \frac{\Delta_{sa}}{K} \mu_k(s,a) \right) - \Delta(s,a) \\ \Delta(s,a) &\leftarrow \left(\sum_{k=1}^{K} \frac{\Delta_{sa}}{K} \mu_k(s,a) \right). \end{aligned} \quad (18)$$

The delta selection then becomes

$$\delta = \begin{cases} \delta_w & if\ \Delta(s,a)\,\Delta^2(s,a) < 0 \\ \delta_l & otherwise \end{cases}. \quad (19)$$

4. The Experimental Domain

Our version of tile world was designed as a test-bed for machine learning methods to be used in the very stochastic world of robot soccer. Specifically, we focus on the learning process for an agent with the ball to decide which team mate to pass it to, and for the team mates to decide where to move to best facilitate moving the ball towards the goal.

The domain for this experiment is the modified TileWorld domain [8] used by Berenji and Vengerov [4,5]. It consists of a 20 x 20 grid world which contains 5 reward opportunities and 5 deformations. The reward opportunities each have random value of 20 to 100 points and a random life span of 5 to 15 time steps. Anytime the agent reaches a reward or the reward expires, it disappears from the domain and another one is generated elsewhere on the board. Agent can move 1 step each time step.

Each deformation has a random penalty value of -5 to -20 points and, unlike the rewards, these deformations occasionally drift. At each time step each deformation has a 10% chance of moving one square in a random direction. Each deformation is also the center of a potential field that radiates out based on the following equation:

$$P = \frac{v}{(d+1)^2}. \quad (20)$$

Where *v* is the value of the deformation and *d* is the distance from the deformation. The cost of each square in the domain is the sum of the effects of each potential field at that point.

Each state in the domain is represented by 4 state variables:

1. Distance to the reward
2. Value of the reward
3. Estimated life expectancy of the reward
4. Roughness of path to the reward

The distance to the reward is calculated simply using the Pythagorean Theorem. The value of each reward randomly determined at the time it is generated. The estimated life expectancy of a reward (*L*) is calculated by

$$L = m - t(r) \quad (20)$$

where *m* is the mean life span (*m*=10 in this example) and *t(r)* is the number of time steps that reward *r* has existed.

The roughness of the path to the reward is calculated by constructing a rectangle with the agent and the reward at opposite corners. The roughness is the average cost of all the squares in that rectangle.

At each time step the agent must decide which of the reward opportunities to pursue. This decision is based on the state variables described above.

Once the decision is made, the agent moves one square towards that opportunity, the policy is updated and the process repeats.

Because the agent can move in any of eight directions (orthogonally or diagonally) there are always three contiguous squares that the agent can choose from to move towards the selected reward. At each time step the agent simply uses the square with the lowest cost.

With each step, the agent garners a negative reward equivalent to the cost of the square it moves to. The agent only receives a positive reward upon reaching a reward opportunity before it expires.

The function $\mu_k(s,a)$ in equations 5-7, 9, 10, 12, 13, 16 and 18 is the degree of membership of state s to k with respect to action a. The value of each of the state variables is described by 3 fuzzy labels (Small, Medium and Large). The shapes of these fuzzy labels are shown in figure 2.

For each of the state variables, the fuzzy labels are assigned so that they evenly divide the range of possible values for the variable. The degree to which the agent is in one of the fuzzy states is the mean of the degrees to which all the state variables belong to the corresponding

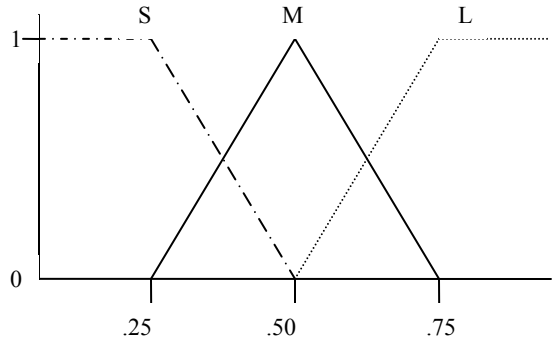

Figure 2: Fuzzy labels used by the agent

labels in the fuzzy state. In this experiment we have used 4 state variables and 3 fuzzy labels resulting in 81 (3^4) total fuzzy states. For comparison purposes, without fuzzy state aggregation, this same domain would have 210 possible distance values, 80 possible reward values, 15 different life expectancy values and at least 1000 different roughness values resulting in 2.52×10^9 possible states. By limiting the state variable values to only integer values (which is not the case in our experiment) this number could be reduced to just over 320,000 states.

At the beginning of each experiment the Q-values are all set to 20. This number is selected because it is comparable to the maximum Q-values found at the end of the experiment and starting with this value results in some natural exploration in the earliest stages of learning. Because the entire $q(k)$ vector and $\pi(k)$ vector are updated at each time step, learning occurs very quickly and no dedicated exploration is required.

5. Experimental Results

Our experiments were conducted by running multiple games of 200 time-steps each. The q and π-vectors were reinitialized at the beginning of each game and the same number of games was run for each algorithm. The parameter settings are $\alpha=0.1$, $\gamma=0.3$, and $\delta=0.5$. Experience shows that the ratio of $\gamma/\alpha=3$ works well in most situations. Increasing the value of α only resulted in a larger magnitude of Q values, but with no corresponding increase in performance. Figure 3 shows the averaged results of running 2000 games with just an on-policy Q-learning algorithm compared to the basic off-policy hill climbing algorithm. The results of multiple games are averaged due to the highly stochastic nature of the domain.

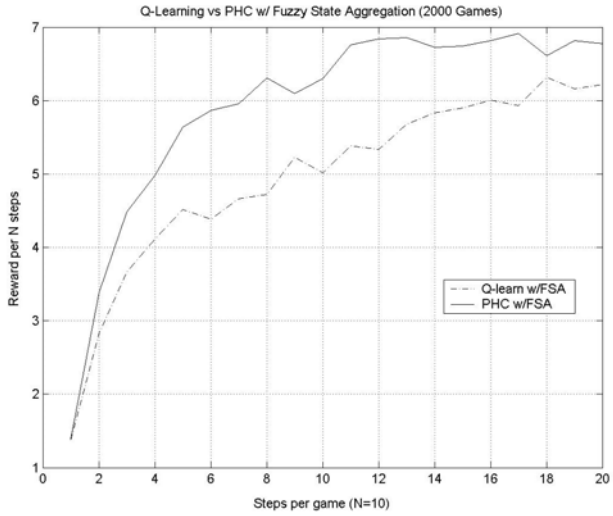

Figure 3: Results of PHC vs. on-Policy Q-Learning

Figure 4 shows results comparing WoLF-PHC and PDWoLF-PHC algorithms to determine if the application of a variable learning rate could improve performance. For these algorithms we used the following parameter settings: $\alpha=0.1$, $\gamma=0.3$, $\delta=0.5$, $\delta_w=0.2$,and $\delta_l=0.8$. These values were chosen for δ_l and δ_w based on Bowling and Veloso's [6] finding that $\delta_l/\delta_w=4$ is a good ratio for using WoLF in stochastic environments.

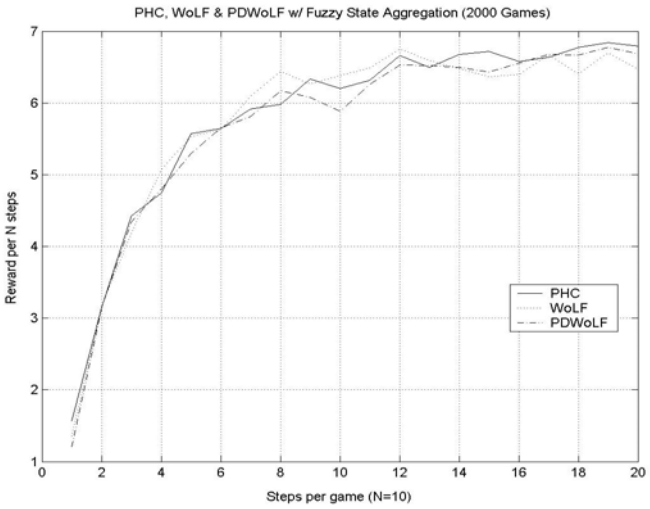

Figure 4: Results of PHC, WoLF and PDWoLF

Our efforts indicate that all three PHC algorithms consistently provide similar results, despite varying the values of each of these variables.

The generalization of the crisp states into a fuzzy state approximation vector smoothes the landscape of the policy table to an extent that the use of the variable learning rate has little effect. The use of the variable learning rate in more chaotic policy landscapes is the key to the improved performance previously demonstrated by the WoLF-PHC and PDWoLF-PHC algorithms.

6. Conclusion and Future Work

This work demonstrates the improvement of combining fuzzy state aggregation (FSA) with each of three different PHC algorithms over standard Q-Learning. Both in terms of speed to convergence and the convergence value itself. The resulting increase in performance clearly shows the benefit of applying the off-policy hill climbing algorithm to the FSA in this highly stochastic environment. Unlike the results of using the WoLF-PHC and PDWoLF-PHC algorithms in a crisp environment, these two algorithms showed no improved performance over the common PHC algorithm.

Our future work will include applying this same combination to more complex domains in an effort to determine if the performance potential of the different algorithms maps to the fuzzy set aggregation function approximation method. We also plan to explore the potential benefit of learning the optimal fuzzy label values for each state variable as a means of further improving performance.

References

[1] R.S. Sutton and A. G. Barto, *Reinforcement learning: an introduction* (Cambridge, Massachusetts, MIT Press, 1998.

[2] S. Lawrence, A.C. Tsoi, and A.D. Back, Function approximation with neural networks and local methods: bias, variance and smoothness, In: P. Bartlett, A. Burkitt and R. Williamson (eds), Australian *Conference on Neural Networks*, Australian National University, Australian National University, 1996, 16-21.

[3] S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In Advances in Neural Information Processing 7, MIT Press, 1994, 361-368.

[4] H.R. Berenji and D. Vengerov, Cooperation and coordination between fuzzy reinforcement learning agents in continuous state partially observable markov decision processes, *Proceedings of 1999 IEEE international Fuzzy Systems Conference*, Seoul, Korea, 1999, 621-627.

[5] H.R. Berenji and D. Vengerov, Advantages of cooperation between reinforcement learning agents in difficult stochastic problems, *Proceedings of the 9th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '00)*, San Antonio, Tx , 2000, 871-876.

[6] M. Bowling and M. Veloso, Multiagent learning using a variable learning rate, *Artificial intelligence 136*, 2002, 215-250

[7] B. Banjeree and J. Peng, Adaptive policy gradient in multiagent learning. *AAMAS '03 International joint conference on Autonomous Agent and Multi- Agent Systems*, Melbourne, Australia, 2003

[8] M.E. Pollack and M. Ringuette, Introducing the tileworld: experimentally evaluating agent architectures, *Eighth National Conference on Artificial Intelligence***,** Menlo Park, CA, 1990.

[9] J.S. Albus, Data storage in the cerebellar model articulation controller (CMAC). *Trans-actions of the ASME: Journal of Dynamic Systems, Measurement, and Control,* 1975, 228-233

[10] L.A. Zadeh, Fuzzy sets. *Journal of information and control 8*, 1965, 338-353.

[11] S.N. Pant and K. E. Holbert. *Fuzzy logic in decision making and signal processing*, http://ceaspub.eas.asu.edu/PowerZone/FuzzyLogic/chapter%202/frame2.htm, March 2004.

[12] C. J. C. H. Watkins, *Learning from delayed rewards*, Cambridge, UK, Cambridge University, Ph.D. thesis, 1989.