International Journal of Computational Intelligence and Applications
© World Scientific Publishing Company

FUZZY STATE AGGREGATION AND POLICY HILL CLIMBING FOR STOCHASTIC ENVIRONMENTS

Dean C. Wardell and Gilbert L. Peterson
Department of Electrical and Computer Engineering, Air Force Institute of Technology, 2950 Hobson Way
Wright-Patterson AFB, OH, 45433, United States

Received (received date)
Revised (revised date)

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 operating environment changes. Additionally, by applying reinforcement learning to multiple cooperative software agents (a multi-agent system) not only allows each individual agent to learn from its own experience, but also opens up the opportunity for the individual agents to learn from the other agents in the system, thus accelerating the rate of learning. This research presents the novel use of fuzzy state aggregation, as the means of function approximation, combined with the fastest policy hill climbing methods of Win or Lose Fast (WoLF) and policy-dynamics based WoLF (PD-WoLF). The combination of fast policy hill climbing and fuzzy state aggregation function approximation is tested in two stochastic environments; Tileworld and the simulated robot soccer domain, RoboCup. The Tileworld results demonstrate that a single agent using the combination of FSA and PHC learns quicker and performs better than combined fuzzy state aggregation and Q-learning reinforcement learning alone. Results from the multi-agent RoboCup domain again illustrate that the policy hill climbing algorithms perform better than Q-learning alone in a multi-agent environment. The learning is further enhanced by allowing the agents to share their experience through weighted strategy sharing.

Keywords: 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 speed 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 the case of multi-agent domains additional progress has been made by allowing the agents to share information and the benefit of their experience via weighted strategy sharing (WSS).[8] Based on the value of their learned information, each agent's learning data is weighted and combined to provide each agent the benefit of information gathered by other agents in the domain.

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[9] environment. The results demonstrate the improved performance of combining an off-policy reinforcement learning method with FSA. The Tileworld tests are followed by experiments in the simulated robot soccer domain RoboCup in which multiple soccer playing agents make up teams and play against each other in a fairly realistic soccer simulation.

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 results of applying the PHC algorithm combined with FSA and WSS in the RoboCup domain comprise the rest of section 4. Section 5 contains a discussion of the conclusions and recommended future research areas.

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 combines 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] 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 0.0 and 1.0.

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.

2.2. *Q-Learning*

In the realm of reinforcement learning, Q-learning[11] 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:

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

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.

2.3. *Fuzzy Logic and Q-learning*

Fuzzy logic has been used to represent continuous state spaces as discrete, thereby making it possible to implement Q-learning in continuous state spaces. The combination of FLCs with Q-learning has been proposed as Fuzzy Q-Learning (FQL) for many single robot applications.[4,5]

When applying Fuzzy Q-Learning to the robot soccer domain it is often in the context of learning one particular skill or behavior. Gu and Hu[12] (and with Specek[13]) present a fuzzy logic controller (FLC) for the implementation of a ball chasing behavior for Sony Aibo robot. Nakashima, et al. propose a fuzzy Q-learning method in which an agent tries to intercept a passed ball based on the inference result by a fuzzy rule based system.[14]

Ammerlaan and Wright[15] address the question of whether systems based on fuzzy logic can effectively adapt themselves to dynamic situations. To answer this question, they design and implement an adaptive fuzzy logic agent for playing RoboCup soccer. The agent has a FLC for basic behaviors, but a neural network allows the agent to adapt to the changes in the environment.

2.4. *Combining fuzzy state aggregation with Q-learning*

In a domain with a large state-space, it is inefficient to learn separate Q-values for each state-action pair. Therefore, it is 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]

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

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

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

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

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

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:

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

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:

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

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. As shown in table 1, 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 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.

Table 1: Basic Policy Hill Climbing

```
Input $\alpha \in (0,1], \quad \delta \in (0,1]$.
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
      $\Delta_{sa} = \begin{cases} -\delta_{sa} & if\ a \neq \max_a Q(s,a') \\ \sum_{a' \neq a \in A} \delta_{sa} & otherwise \end{cases}$
    where $\delta_{sa} = \min\left(\pi(s,a),\ \frac{\delta}{|A|-1}\right)$
Update $\pi(s,a) \leftarrow \pi(s,a) + \Delta_{sa}$
```

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 shown in table 2 are used to calculate δ for the WoLF-PHC algorithm, and are the only changes to PHC as described in Table 1.

Table 2: Additional functions for WoLF-PHC

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

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

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 Δ*(s,a)* with the change in policy from the current time step Δ^2*(s,a)*, shown below.

Table 3: Additional functions used in PDWoLF-PHC

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

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

---

3. Combining PHC and FSA

The combination of PHC and fuzzy state aggregation begins with the state-space constrained to K total fuzzy states. We apply three different variants of a Policy Hill Climbing algorithm; standard PHC, Win or Lose Fast (WoLF) PHC and Policy Dynamics (PD) WoLF-PHC. The implementations of these algorithms use two vectors representing the learned parameter data. The q-vector *q(k)* as described previously and a policy vector π*(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 π*(k)* are initialized as shown below:

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

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

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

Normalizing it with a Boltzmann distribution to ensure (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:

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

Where

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

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

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

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

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 *π(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:

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

This vector is updated by

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

where C is a counting function which tracks 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:

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

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 (10) and is derived from the δ calculation of WoLF in equation 16.

3.2. *Combining PDWoLF and Fuzzy State Aggregation*

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

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

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

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

The delta selection then becomes

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

4. The Experimental Domain

4.1. *Tileworld*

We are interested in developing learning processes for an agent with the ball to decide which team mate to pass it to, which direction to dribble or where to aim when making a goal shot. As a preliminary test, the very stochastic Tileworld[8] domain is used as a proof of concept.

Specifically, this is the modified Tileworld domain used by Berenji and Vengerov,[4,5] consisting 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. The 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:

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

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 - calculated as the Euclidean distance

(2) Value of the reward - from the initial generation
(3) Estimated life expectancy of the reward *(*t) - calculated as L=*m-t(r)* 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.
(4) Roughness of path to the reward - 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.

The value of each of the state variables is described by 3 fuzzy labels Small, Medium and Large at 0.25, 0.50, and 0.75 respectively.

At each time step the agent must decide which of the reward opportunities to pursue, based on the state variables described above. Once the decision is made, the agent moves one square (orthogonally or diagonally) towards that opportunity, the policy is updated and the process repeats.

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.

In this experiment the 4 state variables and 3 fuzzy labels result 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 2.52x10^9 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 π*(k)* vector are updated at each time step, learning occurs very quickly and no dedicated exploration is required.

4.2. *Tileworld 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. For these algorithms we used the following parameter settings: α=*0.1*, γ=*0.3*, δ=*0.5*, δ_w=*0.2*,and δ_l=*0.8* based on guidance given in reference 6. Figure 2 shows the averaged results of running 2000 games with an on-policy Q-learning algorithm, a basic off-policy hill climbing algorithm, WoLF-PHC and PDWoLF-PHC algorithms. The results of multiple games are averaged due to the highly stochastic nature of the domain. As expected, the on-policy Q-learning algorithm performs slightly worse than the policy hill climbing algorithms. Of interest is the fact that all three PHC algorithms consistently provide similar results.

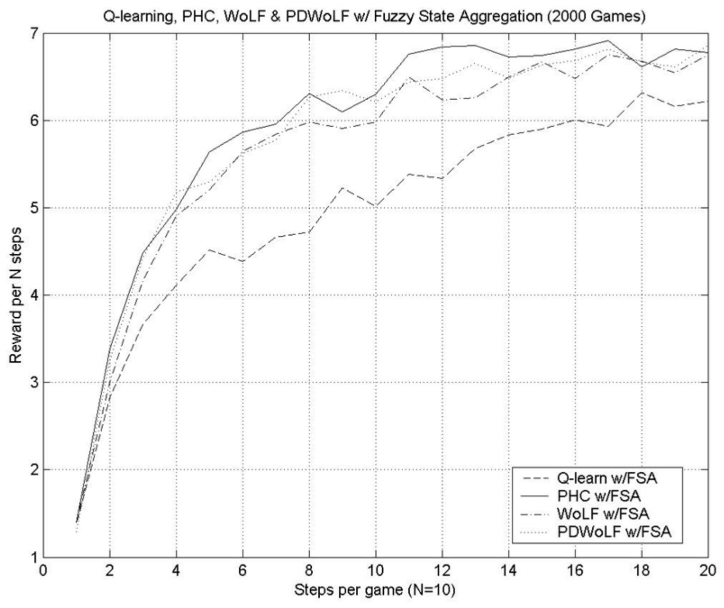

Fig. 1. Results of PHC vs. on-Policy Q-Learning

The generalization of the crisp states into a fuzzy state approximation vector smoothes the landscape of the policy table to the 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.

4.3. *RoboCup Multi-Agent Experiments*

In the RoboCup domain, each agent learns a good policy for which action to take when it has possession of the ball. The decision space consists of three possible actions; shoot for a goal, pass the ball or dribble it. This decision is based on which of these actions has the highest expected reward for the current state of the game. The expected value for each of these actions is calculated using FSA and a standard PHC algorithm.

Passing the Ball - To calculate the expected reward for passing the ball, the agent with the ball considers each teammate (except the goalie) that is visible and to which the agent has a clear line of sight. For each teammate, the state variables the agent uses are

- How many opponents are around the teammate
- How far away is the teammate

- How much closer to, or further from, the opponent's goal is that teammate

The number of Opponents around the teammate is calculated by projecting a 45^o cone from the agent to the teammate and counting the number of opponent players located inside that cone.

The distance to the teammate is the Euclidean distance, and the relative closeness to the opponent goal is the difference between the players' X positions.

As in the tile world experiment, fuzzy state aggregation constrains the number of states in the domain. There are three fuzzy labels (sets) for each state variable resulting in 3^3 or 27 total fuzzy states used for passing the ball. Since the RoboCup domain is continuous, the actual number of states is incalculable.

Dribble the Ball - In calculating the expected reward of dribbling the ball, the agent considers dribbling in each of eight different directions. The eight directions are 45^o apart in a complete circle around the agent, beginning with the -Y direction. For each possible direction the agent uses two state variables

- Number of opponents in that direction
- Degrees away from a direct path to the goal

The number of opponents in the direction is calculated by projecting a 45^ocone to a point 10 meters away in that direction and counting the opponents within that cone. The second state variable value is simply the difference between the direction in question and the angle to the goal. Using fuzzy state aggregation with three fuzzy labels and these two state variables the domain consists of nine (3^2) fuzzy states for dribbling the ball.

Shots on Goal - To calculate the expected reward of shooting the ball, the agent considers shooting at each of seven different points in the goal. The seven points are all along the back of goal, starting at dead center (55, 0) and working out 2 meters at a time. For each possible target point the agent uses three state variables

- Number of opponents along the path to the target point
- Distance between the target point and the opposing goalie
- Distance to that point

The number of opponents near the path to the target point is calculated by projecting a 15^o cone from the agent to the target point and counting the number of opponent players located inside that cone, as shown in figure 3.9. The distance between the goalie and the target point is simply the difference in Y value of the goalie's location compared to the Y value of the target point. The distance to the target point is a simple Euclidean distance.

Using fuzzy state aggregation with three fuzzy labels and these two state variables the domain is constrained to 27 (3^3) fuzzy states for shots on goal.

After making the decision, the agent receives feedback in the form of a penalty or reward based on the outcome of that decision. This feedback is used to update the policy for that action.

4.4. *Weighted Strategy Sharing*

Because each agent runs this algorithm independently of its teammates, each agent learns a different policy based on its individual experience. Weighted Strategy Sharing (WSS) allows the agents to benefit from their teammates' experiences and develop more refined policies. The communication available between agents during game play is limited in range, constrained in its content size and not completely reliable. For all of these reasons, and the fact that RoboCup rules prohibit any inter-agent communication outside the simulator, the WSS occurs off-line at the end of each game.

At the conclusion of the game, each agent stores their respective Q-vector and total reward for each of the three activities using reinforcement learning (passing, dribbling and shooting). The WSS algorithm accesses these files and creates a single updated vector using the "Learning From All" [8] weight assignment mechanism in which agent j is weighted by agent i using the equation:

$$W_i j = \frac{e_j}{\sum\limits_{k=1}^{n} e_k} \tag{21}$$

Where n is the total number of agents and e_k is the amount of the expertness of agent k. The individual expertness value for each agent is calculated using the absolute value of the algebraic sum of the reinforcement signals (r_i).

$$e_i = \left|\sum r_i\right| \tag{22}$$

The updated Q vector is used as the initial vector for the next game, and is calculated as follows:

$$Q_i^{new} \leftarrow \sum_{j=1}^{n} (W_{ij} * Q_j^{old}). \tag{23}$$

4.5. *RoboCup Experimental Results*

To obtain a quick snapshot of how well the algorithm works in the RoboCup soccer domain, the team using FSA and a PHC play against a "Dummy" team. The Dummy team is simply an earlier version of the experimental team which uses conventional if–else statements in selecting the action taken.

From the perspective of the learning agent, opponent players are treated as part of the environment. The Dummy team generally performs poorly, but provides a good stochastic environment in which to measure the performance of the experimental team. The parameter settings are; α*=0.1,* γ*=0.3,* δ*=0.5.* Figure 3 shows the evaluation of the learning ability of the experimental team using Q-learning with FSA, and PHC with FSA against the Dummy team. These results are the average reinforcement points collected by the team each time the agents had control of the ball. Each curve is a collective average over five runs of 10 games each. The average

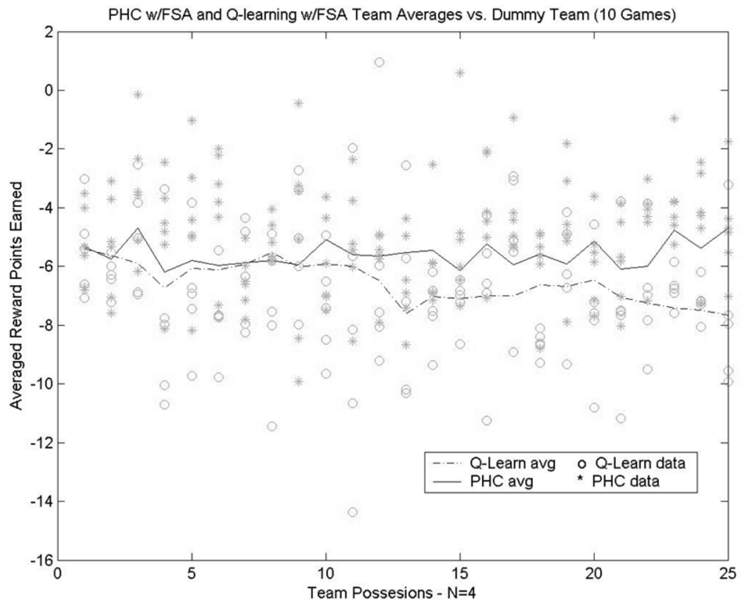

Fig. 2. Q-learning with FSA and PHC with FSA Performance vs.the Dummy Team

lines plotted through the data points are based on the average value of the team's reward points at that time.

Consistent with results from the Tileworld experiment, the PHC algorithm demonstrates a better learning ability over the Q-learning algorithm, despite the fact that both algorithms use the exact same FSA method.

4.6. *Results using Weighted Strategy Sharing*

Because multiple agents are learning in the environment simultaneously, it is beneficial for them to share what they learn. The inclusion of WSS implemented at the end of the game provides each agent with a new Q-vector for the next game based on the inputs of the other team members.

The results in Figure 4 are a comparison of the evaluated learning ability of the experimental team with and without weighted strategy sharing. As anticipated, the experimental team shows an increased learning ability using WSS over the course of these games.

Clearly, the use of weighted strategy sharing increases the rate of learning over that of agents independently learning at their own rate.

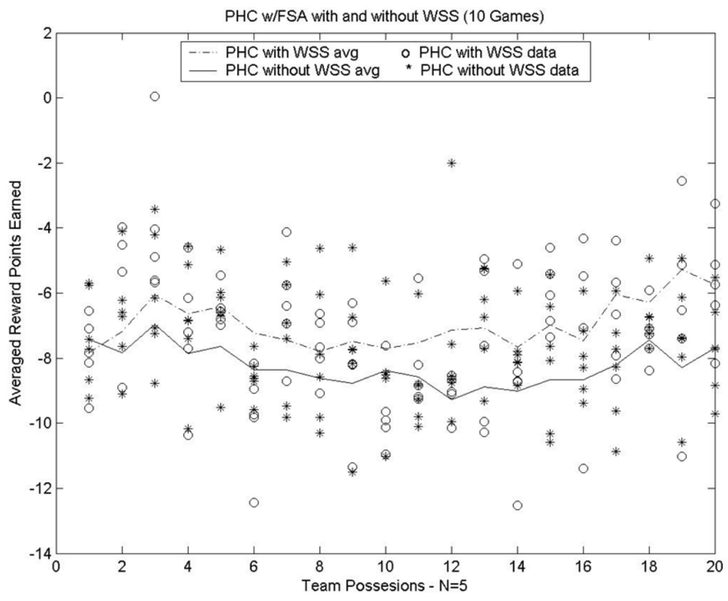

Fig. 3. PHC with FSA Games With and Without WSS

5. 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.

ACKNOWLEDGMENT

The authors wish to thank representatives of the United States Air Force Research Laboratory (AFRL) for their professional interest and support of this research effort.

The views expressed in this paper are those of the authors and do not endorse, or reflect the official policy or position of, the United States Air Force, Department of Defense, or the United States Government.

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. *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.
9. M.E. Pollack and M. Ringuette, Introducing the tileworld: experimentally evaluating agent architectures, *Eighth National Conference on Artificial Intelligence*, Menlo Park, CA, 1990.
10. L.A. Zadeh, Fuzzy sets. *Journal of information and control 8*, 1965, 338-353.
11. C. J. C. H. Watkins, *Learning from delayed rewards*, Cambridge, UK, Cambridge University, Ph.D. thesis, 1989.
12. D. Gu and H. Hu. "Reinforcement Learning of Fuzzy Logic Controllers for Quadruped Walking Robots," *Proceedings of the 15th IFAC World Congress*, Barcelona, Spain, July 21-26, 2002.
13. D. Gu, H. Hu & L. Spacek. "Learning Fuzzy Logic Controller for Reactive Robot Behaviours," *International Conference on Advanced Intelligent Mechatronics* (AIM 2003), Kobe, Japan, July 20-24, 2003.
14. T. Nakashima, M. Udo, and H. Ishibuchi. "Acquiring the Positioning Skill in a Soccer Game using a Fuzzy Q-Learning," *Proceedings of 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1488-1491, July 16-20, Kobe, Japan, 2003.
15. J. Ammerlaan & D. Wright. "Adaptive Cooperative Fuzzy Logic Controller," *Computer Science 2004 Proceedings of the ACS Conferences in Research and Practice in Information Technology (CRPIT) 26*, 2004.