2007-09-27

Limitations and Extensions of the WoLF-PHC Algorithm

Philip R. Cook
Brigham Young University - Provo

LIMITATIONS AND EXTENSIONS OF THE WOLF-PHC ALGORITHM

by

Philip R. Cook

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

December 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Philip R. Cook

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date

Michael A. Goodrich, Chair

Date

Dan Ventura

Date

Scott Woodfield

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Philip R. Cook in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Michael A. Goodrich Chair, Graduate Committee

Accepted for the Department

Parris Egbert
Graduate Coordinator

Accepted for the College

Dr. Thomas W. Sederberg
Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

LIMITATIONS AND EXTENSIONS OF THE WOLF-PHC ALGORITHM

Philip R. Cook

Department of Computer Science

Master of Science

Policy Hill Climbing (PHC) is a reinforcement learning algorithm that extends Q-learning to learn probabilistic policies for multi-agent games. WoLF-PHC extends PHC with the “win or learn fast” principle. A proof that PHC will diverge in self-play when playing Shapley’s game is given, and WoLF-PHC is shown empirically to diverge as well. Various WoLF-PHC based modifications were created, evaluated, and compared in an attempt to obtain convergence to the single shot Nash equilibrium when playing Shapley’s game in self-play without using more information than WoLF-PHC uses. Partial Commitment WoLF-PHC (PCWoLF-PHC), which performs best on Shapley’s game, is tested on other matrix games and shown to produce satisfactory results.

ACKNOWLEDGMENTS

I would like to thank the BYU Computer Science department for funding in the form of teaching and research assistantships. I would like to thank my advisor, Dr. Michael A. Goodrich, for advice, insights, opportunities, his time (and red ink), and for being such a good example of a dedicated scientist and mentor. I thank my family and, most significantly, I thank my parents. They have sacrificed more for me than I realize, and are always willing to do more.

Contents

1 Introduction **1**

1.1 Matrix Games and Strategies 2

1.2 Learning Algorithms 3

1.3 Thesis Statement 7

1.4 Thesis Organization 8

2 Related Literature **9**

2.1 Algorithms 9

2.2 Applications 12

2.3 Cooperation 14

3 Policy Hill-Climbing and Shapley's Game **16**

3.1 Paper-rock-scissors and Shapley's game 18

3.2 Simplifications and their justifications 19

3.2.1 $\gamma = 0$ 19

3.2.2 No More Q-values 20

3.2.3 Learning Rate Observations 22

3.2.4 Iteration grouping 24

3.2.5 Policy calculations 26

3.3 Setup 28

3.3.1 Phase one – t_0 to t_2. 29

3.3.2 Phase two – t_2 to t_4. 30

3.3.3 Phase three – t_4 to t_6. 32

3.3.4 Visualizing a cycle 33

3.3.5 A useful measurement 35

3.3.6 Analysis 38

3.3.7 Independence of Initial Policies 40

3.4 Empirical Results 45

3.4.1 Variables and Equations 46

3.4.2 Parameter Value Observations 46

3.4.3 Decay Parameters and Exploration Strategies. 48

3.4.4 Empirical results for PHC and Shapley's game 49

3.4.5 Empirical results for WoLF-PHC and Shapley's game 52

3.4.6 Pseudoconvergence 54

3.5 Chapter Summary 55

4 Partial Commitment WoLF-PHC (PCWoLF-PHC) 57

4.1 Overview of PCWoLF-PHC 59

4.2 Details of PCWoLF-PHC 71

4.2.1 "Choose Action" 71

4.2.2 "New Highest Reward?" and "Reset Q-values and Minima" 73

4.2.3 "Taking Too Long?" and "Grow Simplex" 74

4.2.4 "Interpolate Q-values and Record New Minima" 74

4.2.5 "Time to Move Simplex?" 75

4.2.6 "Shrink and Move Simplex" 75

4.3 PCWoLF-PHC Simulations 76

4.4 Chapter Summary 78

5 Other Games 79

5.1 Matrix Games 79

5.2 Simulations 84

5.2.1 Three-Person Matching Pennies 85

5.2.2 Fighters and Bombers [15] 85

5.2.3 Battle of the Sexes 86

5.2.4 Tricky Game [9] 86

5.2.5 Modified Shapley's Game 86

5.2.6 Off-center Shapley's Game 87

5.2.7 Three-Person Shapley's Game 87

5.3 Chapter Summary 88

6 Conclusions 90

6.1 Modifications 90

6.2 Future Work 91

A Validation of learning rate equations 102

B WoLF-PHC Modifications, In Detail 105

B.1 Weighted WoLF-PHC (WWoLF-PHC) 107

B.2 Voronoi WoLF-PHC (VWoLF-PHC) 110

B.3 Consecutive WoLF-PHC (CWoLF-PHC) 113

B.4 Reward Penalizing WoLF-PHC (RPWoLF-PHC) 114

B.5 Policy Penalizing WoLF-PHC (PWoLF-PHC) 118

C Parameters Used in PCWoLF-PHC Simulations 126

List of Tables

1.1 Q-learning algorithm. 4

1.2 Policy hill-climbing algorithm. 5

1.3 WoLF-PHC algorithm. 6

3.1 Policy hill-climbing algorithm. 17

3.2 Subset of values used in simulations. 49

4.1 WoLF-PHC modifications. 58

4.2 PCWoLF-PHC algorithm - 1. 65

4.3 PCWoLF-PHC algorithm - 2. 66

4.4 MoveSimplex(s) method of PCWoLF-PHC. 67

4.5 PCWoLF-PHC parameters - 1. 68

4.6 PCWoLF-PHC parameters - 2. 69

4.7 PCWoLF-PHC parameters - 3. 70

B.1 WoLF-PHC modifications. 106

B.2 Weighted WoLF-PHC algorithm. 108

B.3 Voronoi WoLF-PHC algorithm. 111

B.4 Consecutive WoLF-PHC algorithm. 115

B.5 Reward Penalizing WoLF-PHC algorithm. 119

B.6 Policy Penalizing WoLF-PHC algorithm. 123

List of Figures

1.1 Some Sample Matrix Games. 2

3.1 Paper-Rock-Scissors and Shapley's game. 18

3.2 Best response dynamics for Paper-Rock-Scissors. 19

3.3 Circular shift. 21

3.4 Information passing in PHC. 23

3.5 Three-action probability simplex. 34

3.6 Probability simplex with policies and histories. 35

3.7 Pieces of triangle measure. 36

3.8 Triangle measure. 37

3.9 Triangle measure example. 38

3.10 Voronoi cells. 41

3.11 Jordan's three-person matching pennies. 47

3.12 WoLF-PHC and Jordan's three-person matching pennies. 48

3.13 Epsilon-greedy exploration. 50

3.14 PHC and Shapley's game. 51

3.15 WoLF-PHC and Shapley's game. 53

3.16 Identity game. 55

3.17 Pseudoconvergence. 56

4.1 Coordination game and Battle of the Sexes. 60

4.2 Information passing in PCWoLF-PHC. 62

4.3 Flow of control in PCWoLF-PHC. 63

4.4 Three simplices. 64
4.5 Picking an action in PCWoLF-PHC. 72
4.6 PCWoLF-PHC and Shapley's game. 77

5.1 Matrix games to test PCWoLF-PHC. 80
5.2 WoLF-PHC and Jordan's three-person matching pennies. 81
5.3 WoLF-PHC and various matrix games. 82
5.4 PCWoLF-PHC and Jordan's three-person matching pennies. 85
5.5 PCWOLF-PHC and Tricky game. 86
5.6 PCWoLF-PHC and Modified Shapley's game. 87
5.7 PCWoLF-PHC and off-center Shapley's game. 88
5.8 PCWoLF-PHC and three-person Shapley's game. 89

B.1 WWoLF-PHC simulations. 109
B.2 VWoLF-PHC simulations. 112
B.3 CWoLF-PHC simulations - 1. 116
B.4 CWoLF-PHC simulations - 2. 117
B.5 RPWoLF-PHC simulations - 1. 120
B.6 RPWoLF-PHC simulations - 2. 121
B.7 PWoLF-PHC simulations - 1. 124
B.8 PWoLF-PHC simulations - 2. 125

Chapter 1

Introduction

Multiagent systems research is a growing subfield of Artificial Intelligence. Multiagent systems, as defined by Michael Wooldridge [54], are "systems in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks." Multiagent systems could be useful in a number of problems, such as computer games, security, e-finance, auctions, and the stock market.

In a multiagent system where an optimal strategy is not apparent, learning is one way to decide how to make agents act. Unfortunately, learning in multiagent systems is more difficult than with only one agent. When all of the agents are learning, an agent must learn a "moving target." Reinforcement learning algorithms are a common solution to this problem in the computer science literature.

In the literature, two main classes of multiagent learning algorithms are used. They are equilibrium learners, which try to play a Nash equilibrium, and best response learners, which try to play the strategy that will maximize average reward. Q-learning based approaches are best response algorithms and will be the main focus of this thesis.

This thesis will apply variants of policy hill climbing [9], a Q-learning-based best response learning algorithm, to matrix games in self-play. The main goal is to improve upon existing policy hill climbing algorithms and cause them to learn an equilibrium in self-play on a larger class of matrix games.

$$\begin{bmatrix} (0,0) & (1,0) & (0,1) \\ (0,1) & (0,0) & (1,0) \\ (1,0) & (0,1) & (0,0) \end{bmatrix}$$

(a) Shapley's Game

$$\begin{bmatrix} (3,3) & (1,4) \\ (4,1) & (2,2) \end{bmatrix}$$

(b) Prisoner's Dilemma

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

(c) Matching Pennies

(Payoffs for Player 1)

Figure 1.1: Some Sample Matrix Games.

1.1 Matrix Games and Strategies

Bowling and Veloso [9] give the definition of a *matrix game* as the following:

"A *matrix game* or *strategy game* is a tuple $(n, \mathcal{A}_{1...n}, R_{1...n})$ where n is the number of players, $\mathcal{A}_i$ is the set of actions available to player i (and $\mathcal{A}$ is the joint action space $\mathcal{A}_1 \times ... \times \mathcal{A}_n$), and R_i is player i's payoff function $\mathcal{A} \rightarrow \Re$. The players select actions from their available set and receive a payoff that depends on *all* the players' actions. These are often called matrix games, since the R_i functions can be written as n-dimensional matrices."

Writing the reward functions as matrices and given the actions $a_1 \in A_1, ..., a_n \in A_n$, player i would receive a payoff of $R_i(a_1, ..., a_n)$, where $R_i(a_1, ..., a_n)$ is the value in player i's matrix corresponding to the combination of actions played. Figure 1.1 has some example matrix games. Two-player zero-sum matrix games, defined shortly, can be defined as one matrix since one player's loss is the other's gain ($R_1 = -R_2$). The other two-player games can be written as one matrix, where the elements are tuples showing payoffs for all agents. Player 1 would get the first element in the tuple, etc. We will follow a convention where player 1 is the row agent (i.e., player 1's actions decides what row the payoffs will come from), and player 2 is the column agent. If there are more than 2 agents, then multiple matrices are needed.

Constant sum games, where the reward received by all players always adds up to the same constant, are completely competitive. Zero-sum games are well studied, and are a special case of constant sum games where the sum is zero for every outcome. Matching pennies is zero-sum, so in Figure 1.1 only the payoffs for player 1 are listed for that game. Every matrix game is a general sum game, and constant sum games are special cases of general sum games. General sum games can range from being completely cooperative to completely competitive.

Shapley's game is one interesting matrix game. It is given in Figure 1.1a. It is a simple example of a game that can maximize the number of actions that are played before an action-pair is repeated when the agents use a best response dynamic. To see this, assume the players would always change their actions to the best response to the other player's action. Shapley's game would then take the longest possible amount of time, for a three-by-three matrix game, before the players repeat their actions. Shapley's game is of particular interest in this thesis.

An agent may have a policy that is to play a certain action every time. This type of policy is called a *pure strategy.* A *mixed strategy* is more general and chooses actions based on a probability distribution function, where a given action will be randomly chosen a certain percentage of the time.

A *best response* to the opponents' policies is a policy that results in the highest average reward possible to receive against that set of policies. A *Nash equilibrium* occurs when every agent is playing a best response to every other agent. In other words, if a set of agents have policies that are in a Nash equilibrium, then no agent has any incentive to unilaterally change their strategy. It has been proven that every matrix game has at least one Nash equilibrium [31].

1.2 Learning Algorithms

The main learning algorithms studied in this thesis are ones based on Q-learning [48]. Q-learning is a reinforcement learning algorithm that does not need a model of its environment. It learns a policy by selecting actions and modifying an internal value structure

```
1. Let $\alpha$ be a learning rate, and $\gamma$ be the discount factor for expected rewards.
   Initialize, $Q(s,a) \leftarrow 0$.
2. Repeat,
   (a) From state $s$ select an action $a$.
   (b) Observing reward $r$ and next state $s'$.
       $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left( r + \gamma \max_{a'} Q(s',a') \right)$
```

Table 1.1: Q-learning algorithm [48]

to remember the consequences. The algorithm for Q-learning is shown in Table 1.1.

Q-learning keeps track of the "quality" of performing each action in each state. This quality is encoded as a Q-value, which represents both immediate reward and the expected discounted reward for choosing optimally thereafter. The Q-learning algorithm works by keeping track of rewards received by taking actions and then modifying the Q-value associated with that state-action pair. The Q-value is updated by performing a linear combination of the old value and the sum of the immediate reward received and the expected future reward.

Though not explicitly stated in Table 1.1, α must be decayed as the algorithm progresses. Without α decay, Q-values would continue to bounce around a value but never converge to it.

C. Watkins proved that the Q-learning algorithm will always converge to the correct Q-value of the state, given certain conditions [48]. First, the world must be Markovian; the rewards received and subsequent state obtained are based only upon the present state and the action, and not on previous states in the history. Second, the learning rate must decrease fast ($\exists C \in \Re, \lim_{i\to\infty} \sum_i \alpha_i^2 \leq C$) but not too fast ($\forall C \in \Re, \lim_{i\to\infty} \sum_i \alpha_i > C$). Third, the environment must be stationary. Lastly, the agent must visit each state-action pair an infinite number of times. In other words, for each pair (s,a) the agent needs to

1. Let α and δ be learning rates. Initialize,

   $Q(s,a) \leftarrow 0, \quad \pi(s,a) \leftarrow \frac{1}{|A_i|}$

2. Repeat,

   (a) From state s select action a with probability $\pi(s,a)$, with suitable exploration

   (b) Observing reward r and next state s'.

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

   (c) Update $\pi(s,a)$ and constrain it to a legal probability distribution,

   $$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \arg\max_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}$$

Table 1.2: Policy hill-climbing algorithm (PHC) for player i [9].

choose action a in state s an infinite number of times. Learning in multiagent systems breaks the third condition, because all agents are simultaneously learning which causes the Q-learner to try to “hit a moving target.”

The policies that the Q-learning algorithm learns are always pure strategies. The action learned is the one that maximizes the Q-value at that state. Policy Hill Climbing (PHC) [9] adds the ability to learn mixed strategies to Q-learning. The policy consists of a probability distribution function that is modified as the agent takes actions. The algorithm for PHC is given in Table 1.2.

PHC updates a mixed strategy by giving more weight to the action that the Q-learning layer believes is best. It does this by subtracting an equal amount from all the other actions, and adding what was subtracted to the action specified by the maximum Q-value.

Bowling and Veloso extended this with the WoLF principle [9]. WoLF stands for “Win or Learn Fast.” The idea behind WoLF is to quickly adapt when losing but be cautious

1. Let $\alpha, \delta_l > \delta_w$ be learning rates. Initialize,

$$Q(s,a) \leftarrow 0, \quad \pi(s,a) \leftarrow \frac{1}{|A_i|}, \quad C(s) \leftarrow 0.$$

2. Repeat,

   (a) From state s select action a with probability $\pi(s,a)$, with suitable exploration

   (b) Observing reward r and next state s'.

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

   (c) Update estimate of average policy, $\bar{\pi}$,

$$C(s) \leftarrow C(s) + 1$$

$$\forall a' \in A_i \quad \bar{\pi}(s,a') \leftarrow \bar{\pi}(s,a') + \frac{1}{C(s)}(\pi(s,a') - \bar{\pi}(s,a')).$$

   (d) Step π closer to the optimal policy w.r.t. Q. Same as PHC (2c), but with

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

Table 1.3: WoLF-PHC algorithm for player i [9].

when winning. PHC extended with the WoLF principle is aptly known as WoLF-PHC. The algorithm for WoLF-PHC is shown in Table 1.3.

WoLF-PHC works by keeping track of an average policy, $\bar{\pi}(s, a)$, and comparing the current policy, $\pi(s, a)$, to the average. The comparison is done by using the current Q-values to determine (a) the average reward for using the current policy and (b) the average reward for using the average policy. If the calculated value for the current policy's average reward is greater than the value for the average policy, then the agent is "winning" and will use the winning delta value to slowly update the policy. Otherwise the agent is "losing" and will use the corresponding delta value to try and adapt more quickly.

The WoLF principle can be thought of as implementing staggered learning in an intelligent manner. Staggered learning is when you hold the policies of all agents stationary while one agent learns for a while and then choose another agent to be the only one to learn, and so on [50]. This helps lessen the impact of the non-stationary world caused by learning in the presence of other learning agents.

We introduce the term *relatively stationary* and suggest that this concept may be why WoLF is so powerful. An agent is *relatively stationary* to another agent if the other agent's policy is changing fast enough that the first agent's policy seems to be static. WoLF-PHC accomplishes this through the use of the two different learning rates. It effectively changes multiagent learning into learning for short periods in numerous similar stationary worlds.

1.3 Thesis Statement

In this thesis we will show empirically and mathematically that PHC will not converge to the Nash equilibrium in self-play when playing Shapley's game. WoLF-PHC will be empirically shown to diverge as well. Then we will analyze WoLF-PHC in self-play, and develop a modification of WoLF-PHC which we will empirically show converges to a Nash Equilibrium on a larger class of games.

1.4 Thesis Organization

In the next chapter we review related literature. Chapter 3 contains the mathematical proof that PHC does not converge in self-play to the Nash equilibrium on Shapley's game. Empirical results for PHC and WoLF-PHC in self-play in Shapley's game are also given. Chapter 4 discusses the PCWoLF-PHC algorithm and presents results from simulations in self-play in Shapley's game. Chapter 5 contains simulations using the PCWoLF-PHC in self-play in seven matrix games other than Shapley's game. We will then conclude and give possible future research directions.

Chapter 2

Related Literature

Many surveys have been written on the subject of multiagent systems. One by Stone and Veloso takes the machine learning perspective [44], while one written by Vidal takes a game theoretic view [47]. These show that there are different ways to think about and solve problems in multiagent systems.

A common framework for simulations in multiagent systems is given by Littman [29]. Markov Decision Processes (MDPs) are often used and Littman gives some simulation results from MDP games. MDPs are simple because consequences from actions only depend on the current state and not on a history.

Partially observable MDPs (POMDPs) are harder than MDPs because the state is not observable. As Anthony Cassandra [14] states, "Although the underlying dynamics of the POMDP are still Markovian, since we have no direct access to the current state, our decisions require keeping track of (possibly) the entire history of the process, making this a non-Markovian process." See [22] for a discussion on POMDPs, and [11], [53], [52], [7], [26], [34], or [27] for examples of algorithms designed for POMDPs.

2.1 Algorithms

Chang and Kaelbling developed a classification system for multiagent algorithms [16]. All multiagent algorithms can be classified by this system by determining what the algorithm believes about the other agent, and how the history of the interaction is stored. For instance, $\mathcal{B}_i \times \mathcal{H}_j$ would mean that the algorithm believes that the opponent's action is based on the last i moves and the algorithm itself uses the last j moves to choose an action. Q-learning, a reinforcement learning algorithm, would be considered $\mathcal{B}_0 \times \mathcal{H}_\infty$. Similarly, policy hill climbing (PHC) and WoLF-PHC, which are based on Q-learning and are discussed in section 1.2, are also classified as $\mathcal{B}_0 \times \mathcal{H}_\infty$ by this system. The classification $\mathcal{B}_0 \times \mathcal{H}_\infty$ means that this algorithm does not believe the opponent uses any history, but everything that happened in the history has an effect on the agent's policy. Additionally, Chang and Kaelbling proposed an algorithm designed to exploit policy hill climbers [16]. According to their classification system, "PHC-Exploiter" would be classified as $\mathcal{B}_t \times \mathcal{H}_\infty$ where t is some finite number.

Banerjee and Peng experimented with how reactivity, or being able to quickly respond to changes after converging, impacts multiagent learning [5]. They looked at policy hill climbers, Q-learners that have been extended to use mixed strategies, and other equally powerful algorithms. They experimented with PHC-Exploiter, which was developed by Chang and Kaelbling [16]. Exploiter takes advantage of Q-learners by bringing the opposing policy to a pure policy, then playing the pure strategy that beats it. It was shown that under certain conditions WoLF-PHC can actually beat Exploiter. The classification system presented by Chang and Kaelbling is examined as well.

IGA (Infinitesimal Gradient Ascent) was created by Sing, Kearns, and Mansour to apply gradient ascent to two-player, two-action games [41]. IGA is similar to PHC, but IGA requires more information. IGA needs to know the payoff matrix and the opponent's strategy. With this information, both agents can determine the current gradient and respond accordingly.

PDWoLF [4] was developed after looking at and modifying the WoLF-IGA algorithm [9], which was based on IGA. Given a matrix game, the WoLF-IGA and PDWoLF algorithms will move through mixed strategy space (probabilistic combinations of actions) using gradient ascent and small movements. PDWoLF's main use is for two-player two-action matrix games. PDWoLF uses a different criterion than WoLF-PHC to determine whether an agent is winning or losing. This criterion is computable for two-player two-action games, whereas the criterion for WoLF-PHC is estimated by using the average policy instead of the optimal policy.

Zinkevich extended IGA beyond two-player two-action games and named the algorithm GIGA (Generalized Infinitesimal Gradient Ascent) [55]. Regret is a measure of exploitability, and Zinkevich proved an upper bound on GIGA's regret. Bowling applied the "Win or Learn Fast" principle to GIGA to create GIGA-WoLF. GIGA-WoLF retains the regret properties of GIGA while making the algorithm "converge in many situations of self-play." Bowling was able to show that GIGA-WoLF guarantees at most zero average regret. GIGA-WoLF does not require knowledge of the other agent's payoffs, unlike WoLF-IGA which used those payoffs to know when it was "winning."

"Multiplicative weights" [21], developed by Freund and Schapire, shows an alternative to Q-learning-like policy updates which are linear combinations of beliefs and rewards. The multiplicative weights algorithm is a modification of the "weighted majority algorithm," which was presented by Littlestone and Warmuth [28] for repeated play games. The multiplicative weights algorithm starts with a mixed policy, and after each iteration it updates its policy using a multiplicative rule.

"AWESOME" [18] is a relatively recent algorithm for repeated play games. It was developed to maximize payoff in repeated play, while trying to avoid being exploited. AWESOME stands for "Adapt When Everybody is Stationary, Otherwise Move to Equilibrium." From the acronym it is apparent that this algorithm requires the ability to precompute an equilibrium, so it must be able to see all of the payoffs ahead of time. It also needs to know when everyone is stationary, and when they are not. If these things are available AWESOME will learn to play optimally against stationary opponents.

2.2 Applications

Bowling and Veloso applied the WoLF principle to an algorithm able to play Goofspiel against itself [9]. Goofspiel is a two-player card game that uses regular face cards. Though Goofspiel is a fairly simple game to explain and play, to play optimally is very difficult because of a very large state space. The algorithm was made using the ideas of the WoLF principle, policy gradient ascent, and tile coding to make it scale well.

Santana, Corruble, and Ratitch applied multiagent reinforcement learning to the problem of patrolling [38]. The definition of patrolling they give is "the act of walking or traveling around an area, at regular intervals, in order to protect or supervise it." To simplify the area being patrolled, they represent it as a weighted graph. They try a number of reinforcement learning techniques, and they all perform reasonably well. Some future work they describe would be to experiment with communication, different reward and state representations, and dynamic environments.

One well studied game is the iterated prisoner's dilemma. The prisoner's dilemma is a two-player two-action matrix game. The payoffs are derived from a situation where two suspects are caught and kept in different rooms. They cannot communicate with each other but must decide whether to deny everything (cooperate with the other suspect) or implicate the other suspect (defect). If they both cooperate with each other, they both get minor charges. If they both defect, they both get major charges. If only one defects, the defector gets off free while the other is maximally penalized. Since the game is iterated, they play the game repeatedly. People usually learn to either always cooperate or always defect [13]. When both agents are using pure strategies of defecting, the result is the unique Nash equilibrium of the stage game (with other Nash equilibria possible in the iterated game). Mutual cooperation results in the highest average payoffs overtime for both agents [2].

Sandholm and Crites [37] developed some Q-learning agents to try and compete with established strategies for playing the iterated prisoner's dilemma. The main established strategy they played against was "Tit-for-Tat," which simply plays whatever the other agent last played. The Q-learners learned to cooperate when playing against Tit-for-Tat, and would most frequently learn to defect when playing against another Q-learner.

Stimpson et al. [43] introduced a multiple agent form of the prisoner's dilemma, which they called the multi-agent social dilemma. They approached this problem using the idea of a satisficing solution [25]. With this, they had success in getting solutions that were close to pareto optimal with high probability, which, for the MASD problem, is similar to everyone cooperating in the prisoner's dilemma.

One interesting problem domain is the area of partially observable Markov decision processes (POMDPs). The difference between POMDPs and MDPs is that the definition of POMDPs includes a set of observations that the agents can experience. The state space is not directly observable and so there can be uncertainty as to the current state. This makes POMDPs more difficult than MDPs because there are frequently fewer observations than there are states. The probability of getting a certain observation in a given state is known. Reinforcement learning techniques in POMDPs associate rewards and penalties with observation-action pairs instead of state-action pairs.

Theodore Perkins developed an algorithm that applies reinforcement learning to POMDPs [34]. He calls the algorithm he presents "Monte-Carlo Exploring Starts algorithm for POMDPs" because it is similar to "Monte-Carlo Exploring Starts algorithm for MDPs" introduced by Sutton and Barto [45]. They are model-free reinforcement learning algorithms that "[maintain] a table of observation-action values, which are updated based on Monte-Carlo samples of the return."

Darius Brazunas and Craig Boutilier used a different approach to solving POMDPs [11]. They implemented a "stochastic local search technique which . . . uses very different heuristics [than gradient ascent] to evaluate moves." They describe their algorithms as a "compromise between full [dynamic programming] and the very restricted form of local search admitted by [gradient ascent]." The algorithm starts by performing "local moves" to build up a finite state controller, and uses a "node tabu list" to help keep the size of the controller small. After a good finite state controller is built, a greedy-like optimization is performed on possible strategies in the finite state controller.

2.3 Cooperation

Claus and Boutilier [17] examined the application of reinforcement learning to cooperative multiagent systems. They explain the difference between independent learners and joint action learners. Independent learners ignore the other agents, whereas joint action learners may use coordination because they observe the other agents' actions. They run simulations to compare various Q-learning algorithms. Classic Q-learning, policy hill climbers, and WoLF-PHC are all independent learners.

Crandall and Goodrich examined how reputation affects human-robot interaction [20] and social dilemmas [19]. Reputation is "what other agents in the society believe that [the agent] will do." Establishing a reputation is one way to help agents receive better payoffs than they would with always playing the Nash equilibrium. In this way it is similar to building trust between agents.

Price and Boutilier [35] examined imitation as an alternative to direct cooperation. They found that allowing agents to observe other agents greatly reduced the time required to learn. They use some strict assumptions but give possible extensions that allow some of these to be relaxed.

Saha, Sen, and Dutta tried extending self-interested agents with future expectations [36]. This promotes cooperation by comparing the cost of helping other agents with the expectation of future interactions. Their problem domain is a "job completion domain." There are a number of agents, and each have jobs to complete. The agents are "experts" in a given job type, and so those jobs take less time and result in higher quality.

A similar paper is one by Sekaran and Sen [39], and uses a package delivery problem as the domain. They also show that agents that refuse to help other agents do worse overall than ones that reciprocate.

Sen also looked at the ability to reveal one's action in the problems of Markovian domains and matrix games [40]. They gathered results from randomly generated matrix games. In games where neither showing your policy nor hiding your policy is dominant, results show that revealing the agent's policy lead to higher payoffs.

Chapter 3

Policy Hill-Climbing and Shapley's Game

Policy hill-climbing (PHC) [9] is simply a layer of complexity added to Q-learning [48]. Instead of using the action with the highest Q-value as the response for a given state, a probabilistic policy π is used which follows a probability distribution function. The probability distribution function consists of one probability per action. Let this policy be expressed as $\pi_i = [\pi(a_1), ..., \pi(a_n)]$, where π_i is the policy for agent i and $\pi(a_j)$ is the probability that action a_j is chosen. This probability distribution function is modified as the agent takes actions. The action with the highest Q-value is played more frequently as it retains its position as the action with the highest Q-value. In other words, the PHC layer steps the current policy toward that pure strategy which the Q-learning layer currently believes is best.

The algorithm for PHC as cited by Bowling and Veloso [9] is given in Table 3.1. In the algorithm, $Q(s, a)$ is the Q-value for taking action a in state s, π is the current policy, $\pi(s, a)$ is the probability of choosing action a in state s according to the current policy, α is the standard Q-learning learning rate, and δ is the step size for changes in the policy's probability distribution function.

In this chapter, we will (a) show theoretically and empirically that PHC does not converge in self-play on Shapley's game, and (b) show empirically that WoLF-PHC does

```
1. Let $\alpha$ and $\delta$ be learning rates. Initialize,
   $Q(s,a) \leftarrow 0, \quad \pi(s,a) \leftarrow \frac{1}{|A_i|}$

2. Repeat,
    (a) From state $s$ select action $a$ with probability $\pi(s,a)$, with suitable exploration
    (b) Observing reward $r$ and next state $s'$,
        $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right)$
    (c) Update $\pi(s,a)$ and constrain it to a legal probability distribution,
        $\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \arg\max_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}$
```

Table 3.1: Policy hill-climbing algorithm (PHC) for agent i [9].

$$\left[\begin{array}{cccc} & P & R & S \\ P & (0,0) & (1,-1) & (-1,1) \\ R & (-1,1) & (0,0) & (1,-1) \\ S & (1,-1) & (-1,1) & (0,0) \end{array}\right]$$

(a) Paper-Rock-Scissors

$$\left[\begin{array}{cccc} & P & R & S \\ P & (0,0) & (1,0) & (0,1) \\ R & (0,1) & (0,0) & (1,0) \\ S & (1,0) & (0,1) & (0,0) \end{array}\right]$$

(b) Shapley's Game

Figure 3.1: Payoffs for Paper-Rock-Scissors and Shapley's game.

not converge either.

3.1 Paper-rock-scissors and Shapley's game

Shapley's game is a matrix game that is conceptually similar to a non-zero sum version of paper-rock-scissors. Like paper-rock-scissors, there are two players and three actions. The payoffs for paper-rock-scissors and Shapley's game are given in Figure 3.1. The actions for both games have been labeled as paper (P), rock (R), and scissors (S) for ease of comparison.

The only difference between paper-rock-scissors and Shapley's game is that the loser is not penalized in Shapley's game. In other words, when a payoff of -1 would have been received in paper-rock-scissors, the payoff is 0 in Shapley's game.

An interesting note is that both games maximize the length of the best response loop for two-player three-action matrix games. In other words, if you look at the best policy you could have against the opponent, and then see what policy would be a best response for that, these games maximize the number of unique policy pairs before a previously seen pair is encountered. Figure 3.2 illustrates the best response dynamics for paper-rock-scissors. The arrows indicate which consequence comes next when using a best response dynamic.

$$\left[\begin{array}{cccc}
 & P & R & S \\
 & & \downarrow & \\
P & (0,0) & (1,-1)\rightarrow & (-1,1) \\
 & & & \downarrow \\
R & \rightarrow(-1,1) & (0,0) & (1,-1)\rightarrow \\
 & \downarrow & & \\
S & (1,-1)\rightarrow & (-1,1) & (0,0) \\
 & & \downarrow &
\end{array}\right]$$

Figure 3.2: Best response dynamics for Paper-Rock-Scissors.

3.2 Simplifications and their justifications

This section examines the case of two agents using PHC when playing Shapley's game against each other. A number of simplifications can be introduced to make proving properties of the algorithm easier. With each simplification, a justification will be given for why it does not invalidate the proof.

3.2.1 $\gamma = 0$

Since Shapley's game is stateless, s' is always the same as s. Therefore $Q(s, a)$ can be replaced by $Q(a)$, effectively ignoring all reference to state,

$$Q(a) \leftarrow (1 - \alpha)Q(a) + \alpha \left(r + \gamma \max_{a'} Q(a') \right).$$

The resulting equation still includes the discounted reward for transitions from the implicit state to itself; this is the term $\gamma \max_{a'} Q(a')$. Including this term simply causes a positive affine transformation when compared to ignoring discounted future reward. You can see this by noticing that, in steady state, the discounted future reward term will always be equal to $\gamma Q(a^*)$, where a^* is the action with the highest Q-value. Removing discounted future reward decreases what the largest Q-value will converge to by a factor of $1 - \gamma$, and leave the relative ordering of Q-values the same. Since discounted future reward will not change what action will have the highest Q-value in steady state, we may remove it yielding

$$Q(a) \leftarrow (1-\alpha)Q(a) + \alpha r.$$

Equivalently, we can set γ to 0. This makes the Q-value update function a simple linear combination of the old Q-value and the reward received. An equivalent formulation is given in [17], and is given below.

$$Q(a) \leftarrow Q(a) + \alpha(r - Q(a)).$$

This simplification will not have an effect on the validity of a proof involving Shapley's game. This is due to the fact that Shapley's game is stateless, and we have shown that the calculations in steady state will be essentially the same.

3.2.2 No More Q-values

Since every time the agents play Shapley's game they choose their action from a probabilistic policy, π, it is helpful to look at average rewards. Consider what happens when an agent using PHC plays Shapley's game against an opponent playing a stationary policy. Given an opposing agent's policy $\pi_2 = [\pi_2(paper), \pi_2(rock), \pi_2(scissors)]$, the average reward for a given action a is

$$\bar{R}_1(a) = \sum_b \pi_2(b) * R_1(a,b), \tag{3.1}$$

where $\bar{R}_1(a)$ is the average reward agent 1 receives when playing a, $\pi_2(b)$ is the probability that agent 2 chooses action b, and $R_1(a,b)$ is the reward agent 1 receives when playing action a given that agent 2 plays action b. $R_1(a,b)$ is taken directly from the payoff matrix. Using Equation 3.1 we can compute $\bar{R}_1(paper)$ as follows

$$\begin{tikzcd}
{} & \pi_2(paper) \arrow[dl] & \pi_2(rock) \arrow[dl] & \pi_2(scissors) \arrow[dl] & {} \arrow[dl] \\
{} & \bar{R}_1(paper) \arrow[d] & \bar{R}_1(rock) \arrow[d] & \bar{R}_1(scissors) \arrow[d] & {} \\
{} & Q_1(paper) & Q_1(rock) & Q_1(scissors) & {}
\end{tikzcd}$$

Figure 3.3: Shows the circular shift from one agent's policy to the other's average reward and Q-values.

$$\begin{aligned}
\bar{R}_1(paper) &= \pi_2(paper) * R_1(paper, paper) \\
&\quad + \pi_2(rock) * R_1(paper, rock) \\
&\quad + \pi_2(scissors) * R_1(paper, scissors) \\
&= \pi_2(paper) * 0 + \pi_2(rock) * 1 + \pi_2(scissors) * 0 \\
&= \pi_2(rock).
\end{aligned}$$

Similarly, $\bar{R}_1(rock) = \pi_2(scissors)$, and $\bar{R}_1(scissors) = \pi_2(paper)$. Therefore, the average reward for an agent's actions can be obtained by looking at the appropriate action's probability in the opponent's policy.

From Watkin's Q-learning convergence proof [49], the Q-value for a given action will approach the average reward as time progresses. So the point of attraction for agent 1's Q-values is agent 2's policy circularly shifted left one action, and the point of attraction for agent 2's Q-values is agent 1's policy circularly shifted left one action. Figure 3.3 shows this circular shift.

Now consider what happens when both agents' policies are being updated using PHC. Let α be small so that the adapting Q-values will not jump around wildly, and choose δ such that $\alpha \gg \delta$ to allow Q-values to approximately converge before policies have a chance to change. Since $\alpha > 0$, the Q-values will not completely converge, but will bounce around very close to their corresponding points of attraction. Since α is small, the policy should not "bounce" much. Using these observations, we can see from the Q-value update equation (2b in Table 3.1) that after a number of iterations, the Q-values will approximately converge to the average rewards for the given actions. Using a notation for

Q-values that is similar to the notation for policies results in

$$Q_i = [Q_i(\mathit{paper}), Q_i(\mathit{rock}), Q_i(\mathit{scissors})].$$

Since $Q \rightarrow \bar{R}$, α is small, and $\alpha \gg \delta$, it follows that

$$\begin{aligned} Q_i &= [Q_i(\mathit{paper}), Q_i(\mathit{rock}), Q_i(\mathit{scissors})] \\ &\approx [\bar{R}_i(\mathit{paper}), \bar{R}_i(\mathit{rock}), \bar{R}_i(\mathit{scissors})] \\ &= [\pi_{!i}(\mathit{rock}), \pi_{!i}(\mathit{scissors}), \pi_{!i}(\mathit{paper})], \end{aligned}$$

where $!i$ represents agent i's opponent.

Thus, the attractor points for Q-values (or the 'true' Q-values) are simply the opponent's policy circularly shifted left. Therefore, the expected Q-values can be identified without actually running the algorithm, which means that we can determine how the agent will learn without actually having them play the game. Since the only randomness is in the agents' action choices when they play the game to determine the Q-values, all randomness can be removed (on average) and expected learning will only depend on initial policies.

This simplification is obviously dependent on α being small and $\alpha \gg \delta$. These assumptions are discussed in Section 3.2.3.

3.2.3 Learning Rate Observations

PHC repeatedly steps the agent's policy toward the pure strategy of deterministically playing the action that currently has the highest Q-value, as determined by the Q-learning layer beneath it. In effect, the Q-learning layer can be thought of as forming a gradient for the PHC layer to climb. Figure 3.4 shows how information is passed in the PHC algorithm. It is intuitive to have α, the learning rate for Q-learning, greater than δ, the gradient step size, to allow the Q-learning layer to adapt more quickly than the PHC layer, and therefore give better gradient information to the PHC layer.

Another reason that α should be greater than δ is found by observing how the learning rates are used. In the Q-learning layer, α is used to determine the percentage that the new reward contributes to the estimate of current Q-value. In the PHC layer, δ is the step size

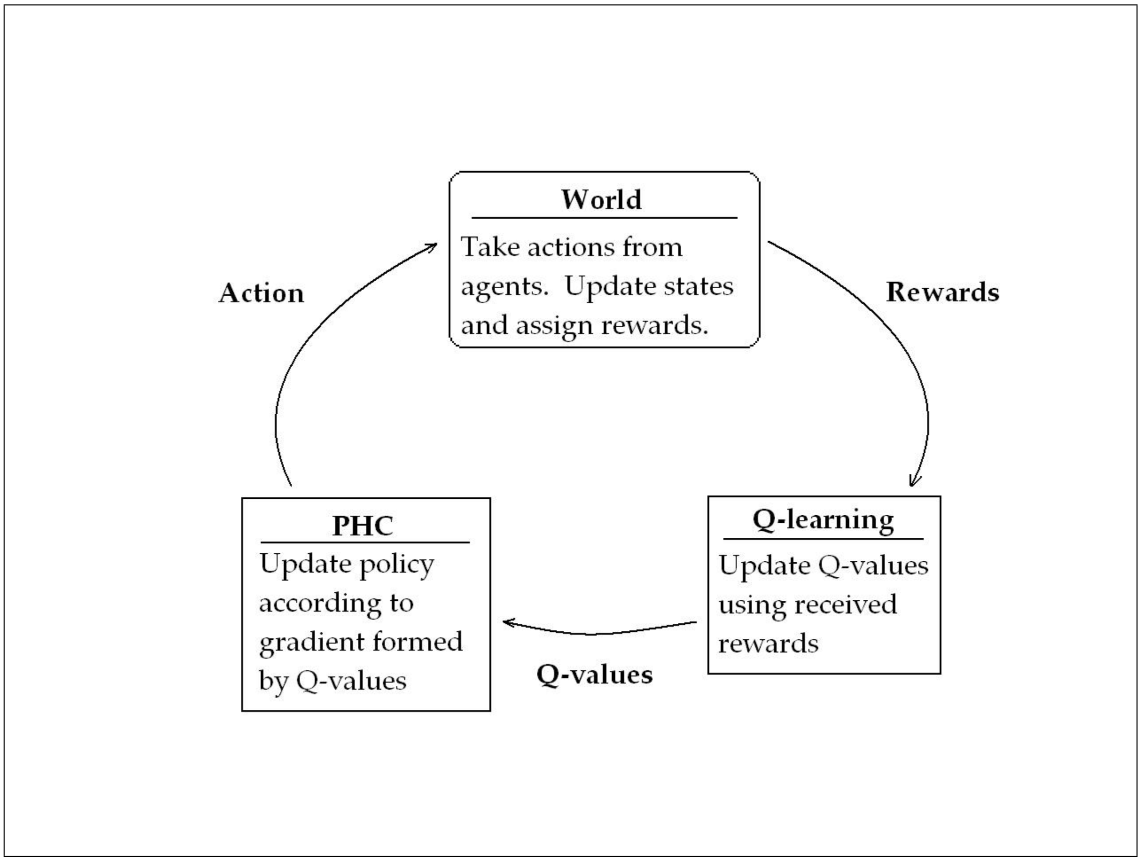
World
Take actions from agents. Update states and assign rewards.
Action
Rewards
PHC
Update policy according to gradient formed by Q-values
Q-learning
Update Q-values using received rewards
Q-values

Figure 3.4: Information passing in the PHC algorithm.

for how far the policy moves towards the action that the Q-learning layer thinks is best. For stateless Q-learning, this is written mathematically as

$$Q(a) \leftarrow (1-\alpha)Q(a) + \alpha R,$$

$$\pi(a^*) \leftarrow \pi(a^*) + \delta,$$

where R is the combination of the current reward and any discounted future rewards, a^* is the action that maximizes $Q(a)$, and a is the action last played by the agent.

Note that given a constant R, $Q(a)$ will approach R asymptotically. $Q(a)$ will move quickly at first, but the closer it gets to R the less movement each update causes. $\pi(a)$, on the other hand, moves a specified amount each update. Thus, it approaches a value linearly which means that the convergence of a small δ is similar to the convergence of a larger α.

Given these observations, we will hereafter assume that α is much larger than δ. Section 3.4.2 gives some experimental justification for this simplification.

3.2.4 Iteration grouping

The update equation (Equation 2c in Table 3.1) for policies needs to find the action with the highest Q-value. Since Q-values approach an attractor that is equivalent to the opponent's policy shifted, and since $\alpha \gg \delta$, the *arg max* component in the update equation in Table 3.1 will yield that action which "wins" against the opponent's most frequently played action. The only time this action will change is when the opponent changes its most preferred action. We will refer to this change as an *arg max shift*.

Consider the following example that shows when an arg max shift occurs. Let agent 1's policy be [.5, .20, .30], and let agent 2 play [0, 0, 1]. In other words, agent 1 plays paper half the time, rock 20% of the time, and scissors 30% of the time while agent 2 always plays scissors. Agent 2 will get a reward of 1 half the time, but agent 1 will only get a reward of 1 20% of the time. Agent 2 will not want to change its policy; the maximizing action is the one it plays deterministically. Agent 1 will want to play rock more because a reward of one is received every time rock is played.

Using Equation 2c in Table 3.1, agent 1's policy after one iteration will be [.5-$\frac{\delta}{2}$, .2+δ, .3-$\frac{\delta}{2}$] and agent 2's policy will remain unchanged. After two iterations, agent 1's policy will be [.5-δ, .2+2δ, .3-δ] and agent 2's policy will remain unchanged. Notice that the amount that the rising action, rock in this case, has increased in a given time period is equal to the sum of the delta values during that time period, so long as an arg max shift did not occur for the agent during that time period.

After playing enough iterations to cause the sum of delta values used to equal 1/15 (approx .0667), agent 1's policy will be about [.4667, .2667, .2667] and agent 2's policy will remain unchanged. Agent 1's policy will have rock and scissors played the same percentage of the time, but since paper is still the most played action no arg max shift occurs.

After playing enough iterations to cause the sum of delta values used since the initial state to equal to 1/5, agent 1's policy will be [.4, .4, .2], and agent 2's policy will have remained unchanged (until possibly the last iteration). Since, for agent 1, the rising action's probability is equal to the greatest falling probability, an arg max shift for agent 2 will have occurred on the current iteration (or will occur on the next iteration). The arg max shift will cause agent 2's policy to start shifting toward favoring paper; agent 1 will play rock most frequently until agent 2 adjusts its policy

This example uses the fact that if an arg max shift does not occur for agent i after n iterations, then agent i's expected policy would have been the same had it only moved one iteration and had a δ value n times as large. This can be seen easily by looking at the update equation when $\arg\max_{a'} Q(s, a')$ and δ are held constant. More specifically, the action with rising probability will increase by an amount equal to the sum of the δ values for the number of iterations completed, and the other two actions will decrease by half that amount each. This is why the example simply used the sum of the delta values for the iterations played. This assumes that the least preferred probability is still non-zero, otherwise the boundary conditions for a probability mass function come into play. To avoid this boundary problem in early calculations, we can simply choose a starting joint policy (both agents' policies) closer to the Nash equilibrium.

Noticing this, we simply need to see how far (in terms of the sum of δ values) policies can move before an arg max shift occurs, then update both agents' policies by that amount. Let this required amount be denoted by δt, where t represents an amount of time. This allows numerous iterations to be effectively grouped together.

Using δt assumes that δ is constant over the time period t. Though not explicitly stated in the algorithm, δ values must be decayed over time to allow convergence. Since we are looking at an amount the policy needs to move, δ decay can be ignored without loss of generality. To include δ decay we simply need to change the interpretation of δt; t would be a set of iterations, and δt would be the sum of the decayed δ values over that set of iterations. With delta decay, $\delta(t_i - t_j)$ would be interpreted as the sum of the delta values from time t_j to time t_i, instead of simply the δ value multiplied by the number of iterations.

This simplification is based solely on the policy update equation, and as such will not affect validity of proofs which are based on it. The example shown does use the idea of removing Q-values as discussed in Section 3.2.2. If Q-values were used, there would simply be a delay while the Q-values adjust to the change in the opponents strategy. This delay would only serve to allow more or different iterations to be grouped together.

3.2.5 Policy calculations

Since an arg max shift is the only event (except boundary conditions) that will prevent iterations from being grouped, we can simplify the process by grouping all iterations between arg max shifts together. We can then simply use these groups to characterize how the agents will learn.

Given policies for two agents, we can determine the δt value required before an arg max shift occurs. Once we identify which agent will cause the arg max shift, δt can be obtained through a simple linear combination of the relevant action probabilities for that agent. We can identify this linear combination, and then use it to update the agents' policies.

The policy update equation increases one action's probability by δ while decreasing the others by $\frac{-\delta}{|A_i|-1}$. Since Shapley's game has three actions for both agents, the amount the two decreasing actions are changed is $-\delta/2$. Let t_0 be the current time and let t_{ams} be the time at which the arg max shift occurs. Let $x(t)$ denote the greatest falling probability at time t and let $y(t)$ denote the rising probability at time t. Equating $x(t_{\text{ams}})$ and $y(t_{\text{ams}})$ yields the following:

$$\begin{aligned} x(t_{\text{ams}}) = x(t_0) - \tfrac{\delta(t_{\text{ams}}-t_0)}{2} &= y(t_0) + \delta(t_{\text{ams}} - t_0) = y(t_{\text{ams}}) \\ x(t_0) - y(t_0) &= \tfrac{3\delta(t_{\text{ams}}-t_0)}{2} \\ \tfrac{2(x(t_0)-y(t_0))}{3} &= \delta(t_{\text{ams}} - t_0) \end{aligned} \tag{3.2}$$

Consequently,

$$\begin{aligned} x(t_{\text{ams}}) = y(t_{\text{ams}}) &= y(t_0) + \delta(t_{\text{ams}} - t_0) \\ &= y(t_0) + \tfrac{2(x(t_0)-y(t_0))}{3} \\ &= \tfrac{3y(t_0)+2x(t_0)-2y(t_0)}{3} \\ &= \tfrac{2x(t_0)+y(t_0)}{3} \end{aligned} \tag{3.3}$$

This simplification is dependent upon (a) the "iteration grouping" simplification, (b) the requirement that $\alpha \gg \delta$, and (c) the requirement that α is small. The requirements that $\alpha \gg \delta$ and that α is small serve only to show that δ is very small and that we can use the opponent's shifted policy instead of calculating Q-values. The larger the δ value is, the larger the distance by which the policies will overshoot the arg max shift after each timestep. This is due to the agent's policy not landing exactly on a strategy that has two actions played the same amount.

In essence, this simplification merely extends the "iteration grouping" simplification to using the maximal iteration grouping. The maximal iteration grouping has separations only when an arg max shift occurs and causes one of the agents to change which action increases in probability. Equations 3.2 and 3.3 assume a small α and that $\alpha \gg \delta$. If those conditions are not met, then an arg max shift is delayed while Q-values adjust to the opponent's strategy.

In the next section, we will break up a cycle into three *phases.* A phase begins with agent 1 causing an arg max shift for agent 2, learning until agent 1 has an arg max shift, and learning again until an arg max shift occurs again for agent 2. Between arg max shifts, we will use Equations 3.2 and 3.3 to update the policies by the required amount. After 3 of these phases, the policies will have taken a full circle. By this we mean that the action's probabilities will be moving the same directions they did at the beginning; i.e., if x and y were falling and z was rising before the cycle, the same pattern will be present after the cycle. We can analyze how far from the Nash equilibrium the agents have travelled over this period, and compare it to where they started. If they are further from the Nash equilibrium, then we can conclude that learning is not converging to the Nash equilibrium.

3.3 Setup

We will now go through the three phases explained in the previous section. Each phase starts and ends with agent 1 causing an arg max shift for agent 2. This means that agent 2 will have one action at its peak probability at the beginning and end of each phase. Without loss of generality, suppose that agent 2's strategy has paper at its peak value. At the end of phase three, agent 2 will again be playing paper at its peak probability for this cycle. After all three phases, both agents' policies will have completed one full cycle.

Let agent 1's initial policy ($\pi_1(t_0)$) be $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$. Let agent 2's initial policy ($\pi_2(t_0)$) be $[\frac{1}{3}+x, \frac{1}{3}+y, \frac{1}{3}+z]$, where x+y+z=0, x>y, x>z, and $-\frac{1}{3} \leq x, y, z \leq \frac{2}{3}$. It does not matter which of y or z is greater.

Since agent 1's policy is to play random, we can see how PHC in self-play will react when getting close to the Nash Equilibrium. We can also think of this as starting at the Nash equilibrium and letting one or two iterations occur. With the randomness in the original algorithm, the agents' policies will move away from the equilibrium a little. Without loss of generality, the action chosen to be the one with the largest probability for agent 2 is paper.

At the beginning of phase one, agent 2's most probable action is paper (the left action).

This will cause agent 1 to change its policy to favor scissors (the right action). Since agent 1's policy will quickly favor scissors, agent 2's policy will change to play rock (the middle action) more. We will follow this process of tracking when arg max shifts occur from this starting point.

3.3.1 Phase one – t_0 to t_2.

The policies at time t_0 are

$$\pi_1(t_0) = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}],$$
$$\pi_2(t_0) = [\frac{1}{3} + x, \frac{1}{3} + y, \frac{1}{3} + z].$$

Without loss of generality, select an initial learning bias for this state and set

$$\pi_1(t_0) = [\frac{1 - \frac{\delta}{2}}{3}, \frac{1 - \frac{\delta}{2}}{3}, \frac{1 + \delta}{3}],$$
$$\pi_2(t_0) = [\frac{1}{3} + x, \frac{1}{3} + y, \frac{1}{3} + z].$$

This change in agent 1's policy is to disambiguate the Q-value maximizing action for agent 2. Since agent 2's most frequently played action is paper, agent 1's policy will increase the probability of scissors. Since agent 1's most frequently played action is quickly going to be scissors, agent 2 will change their policy to play rock more often. Using Equation 3.2 yields

$$\begin{aligned} \delta(t_1 - t_0) &= \tfrac{2((\frac{1}{3}+x)-(\frac{1}{3}+y))}{3} \\ &= \tfrac{2(x-y)}{3} \\ &= \tfrac{2x-2y}{3}. \end{aligned}$$

After being updated by $\delta(t_1 - t_0)$, agent 1's policy is

$$\begin{aligned} \pi_1(t_1) &= [\tfrac{1}{3} - \tfrac{\delta(t_1-t_0)}{2}, \tfrac{1}{3} - \tfrac{\delta(t_1-t_0)}{2}, \tfrac{1}{3} + \delta(t_1 - t_0)] \\ &= [\tfrac{1}{3} - \tfrac{x-y}{3}, \tfrac{1}{3} - \tfrac{x-y}{3}, \tfrac{1}{3} + \tfrac{2x-2y}{3}] \\ &= [\tfrac{1-x+y}{3}, \tfrac{1-x+y}{3}, \tfrac{1+2x-2y}{3}], \end{aligned}$$

and agent 2's policy is,

$$\begin{aligned}\pi_2(t_1) &= [\tfrac{1}{3} + x - \tfrac{\delta(t_1-t_0)}{2}, \tfrac{1}{3} + y + \delta(t_1 - t_0), \tfrac{1}{3} + z - \tfrac{\delta(t_1-t_0)}{2}] \\ &= [\tfrac{1}{3} + x - \tfrac{x-y}{3}, \tfrac{1}{3} + y + \tfrac{2x-2y}{3}, \tfrac{1}{3} + z - \tfrac{x-y}{3}] \\ &= [\tfrac{1+2x+y}{3}, \tfrac{1+2x+y}{3}, \tfrac{1-x+y+3z}{3}].\end{aligned}$$

At time t_1, agent 2 ends up with 2 actions with equal probability. Since paper and rock are played with equal probability and rock is increasing, agent 1 will have an arg max shift occur and will then start to favor playing paper. Agent 1 will be the next to cause an arg max shift for its opponent. The δt value required for this to occur is

$$\begin{aligned}\delta(t_2 - t_1) &= \tfrac{2(\frac{1+2x-2y}{3} - \frac{1-x+y}{3})}{3} \\ &= \tfrac{(2+4x-4y)+(-2+2x-2y)}{3} \\ &= \tfrac{6x-6y}{3} \\ &= \tfrac{2x-2y}{3}.\end{aligned}$$

After being updated by $\delta(t_2 - t_1)$, agent 1's policy is

$$\begin{aligned}\pi_1(t_2) &= [\tfrac{1-x+y}{3} + \tfrac{2x-2y}{3}, \tfrac{1-x+y}{3} - \tfrac{x-y}{3}, \tfrac{1+2x-2y}{3} - \tfrac{2x-2y}{3}] \\ &= [\tfrac{1-x+y+2x-2y}{3}, \tfrac{1-x+y-x+y}{3}, \tfrac{1+2x-2y-x+y}{3}] \\ &= [\tfrac{1+x-y}{3}, \tfrac{1-2x+2y}{3}, \tfrac{1+x-y}{3}],\end{aligned}$$

and agent 2's policy is,

$$\begin{aligned}\pi_2(t_2) &= [\tfrac{1+2x+y}{3} - \tfrac{x-y}{3}, \tfrac{1+2x+y}{3} + \tfrac{2x-2y}{3}, \tfrac{1-x+y+3z}{3} - \tfrac{x-y}{3}] \\ &= [\tfrac{1+2x+y-x+y}{3}, \tfrac{1+2x+y+2x-2y}{3}, \tfrac{1-x+y+3z-x+y}{3}] \\ &= [\tfrac{1+x+2y}{3}, \tfrac{1+4x-y}{3}, \tfrac{1-2x+2y+3z}{3}].\end{aligned}$$

3.3.2 Phase two – t_2 to t_4.

At time t_2, agent 1 will begin to play paper most often. Agent 2 will have an arg max shift and start to play scissors more. Agent 2 will be the next one to cause an arg max shift for the opponent, and this will occur after a δt value of

$$\begin{aligned}
\delta(t_3 - t_2) &= \frac{2(\frac{1+4x-y}{3} - \frac{1-2x+2y+3z}{3})}{3} \\
&= \frac{\frac{2+8x-2y}{3} - \frac{2-4x+4y+6z}{3}}{3} \\
&= \frac{2+8x-2y-2+4x-4y-6z}{9} \\
&= \frac{12x-6y-6z}{9} \\
&= \frac{4x-2y-2z}{3}.
\end{aligned}$$

After being updated by $\delta(t_3 - t_2)$, agent 1's policy is

$$\begin{aligned}
\pi_1(t_3) &= [\frac{1+x-y}{3} + \frac{4x-2y-2z}{3}, \frac{1-2x+2y}{3} - \frac{2x-y-z}{3}, \frac{1+x-y}{3} - \frac{2x-y-z}{3}] \\
&= [\frac{1+x-y+4x-2y-2z}{3}, \frac{1-2x+2y-2x+y+z}{3}, \frac{1+x-y-2x+y+z}{3}] \\
&= [\frac{1+5x-3y-2z}{3}, \frac{1-4x+3y+z}{3}, \frac{1-x+z}{3}],
\end{aligned}$$

and agent 2's policy is,

$$\begin{aligned}
\pi_2(t_3) &= [\frac{1+x+2y}{3} - \frac{2x-y-z}{3}, \frac{1+4x-y}{3} - \frac{2x-y-z}{3}, \frac{1-2x+2y+3z}{3} + \frac{4x-2y-2z}{3}] \\
&= [\frac{1+x+2y-2x+y+z}{3}, \frac{1+4x-y-2x+y+z}{3}, \frac{1-2x+2y+3z+4x-2y-2z}{3}] \\
&= [\frac{1-x+3y+z}{3}, \frac{1+2x+z}{3}, \frac{1+2x+z}{3}].
\end{aligned}$$

At time t_3, agent 2 will end up with 2 actions with equal probability. These will be scissors, which is increasing, and rock, which is decreasing. Agent 1 will then have an arg max shift and start to play rock more often. Agent 1 will be the next to cause an arg max shift, and it will occur after the policies move an amount equal to

$$\begin{aligned}
\delta(t_4 - t_3) &= \frac{2(\frac{1+5x-3y-2z}{3} - \frac{1-4x+3y+z}{3})}{3} \\
&= \frac{\frac{2+10x-6y-4z}{3} - \frac{2-8x+6y+2z}{3}}{3} \\
&= \frac{2+10x-6y-4z-2+8x-6y-2z}{9} \\
&= \frac{18x-12y-6z}{9} \\
&= \frac{6x-4y-2z}{3}.
\end{aligned}$$

After being updated by $\delta(t_4 - t_3)$, agent 1's policy is

$$\begin{aligned}
\pi_1(t_4) &= [\tfrac{1+5x-3y-2z}{3} - \tfrac{3x-2y-z}{3}, \tfrac{1-4x+3y+z}{3} + \tfrac{6x-4y-2z}{3}, \tfrac{1-x+z}{3} - \tfrac{3x-2y-z}{3}] \\
&= [\tfrac{1+5x-3y-2z-3x+2y+z}{3}, \tfrac{1-4x+3y+z+6x-4y-2z}{3}, \tfrac{1-x+z-3x+2y+z}{3}] \\
&= [\tfrac{1+2x-y-z}{3}, \tfrac{1+2x-y-z}{3}, \tfrac{1-4x+2y+2z}{3}],
\end{aligned}$$

and agent 2's policy is,

$$\begin{aligned}
\pi_2(t_4) &= [\tfrac{1-x+3y+z}{3} - \tfrac{3x-2y-z}{3}, \tfrac{1+2x+z}{3} - \tfrac{3x-2y-z}{3}, \tfrac{1+2x+z}{3} + \tfrac{6x-4y-2z}{3}] \\
&= [\tfrac{1-x+3y+z-3x+2y+z}{3}, \tfrac{1+2x+z-3x+2y+z}{3}, \tfrac{1+2x+z+6x-4y-2z}{3}] \\
&= [\tfrac{1-4x+5y+2z}{3}, \tfrac{1-x+2y+2z}{3}, \tfrac{1+8x-4y-z}{3}].
\end{aligned}$$

3.3.3 Phase three – t_4 to t_6.

At time t_4, agent 1 will again end up with two actions taken with equal probability. These will be rock and paper, with rock increasing. Agent 2 will have an arg max shift and start to play paper more. To cause the next arg max shift, agent 2's policy will need to move a δt value equal to

$$\begin{aligned}
\delta(t_5 - t_4) &= \frac{2(\frac{1+8x-4y-z}{3} - \frac{1-4x+5y+2z}{3})}{3} \\
&= \frac{\frac{2+16x-8y-2z}{3} - \frac{2-8x+10y+4z}{3})}{3} \\
&= \frac{2+16x-8y-2z-2+8x-10y-4z}{9} \\
&= \frac{24x-18y-6z}{9} \\
&= \frac{8x-6y-2z}{3}.
\end{aligned}$$

After being updated by $\delta(t_5 - t_4)$, agent 1's policy is

$$\begin{aligned}
\pi_1(t_5) &= [\tfrac{1+2x-y-z}{3} - \tfrac{4x-3y-z}{3}, \tfrac{1+2x-y-z}{3} + \tfrac{8x-6y-2z}{3}, \tfrac{1-4x+2y+2z}{3} - \tfrac{4x-3y-z}{3}] \\
&= [\tfrac{1+2x-y-z-4x+3y+z}{3}, \tfrac{1+2x-y-z+8x-6y-2z}{3}, \tfrac{1-4x+2y+2z-4x+3y+z}{3}] \\
&= [\tfrac{1-2x+2y}{3}, \tfrac{1+10x-7y-3z}{3}, \tfrac{1-8x+5y+3z}{3}],
\end{aligned}$$

and agent 2's policy is,

$$\begin{aligned}\pi_2(t_5) &= [\tfrac{1-4x+5y+2z}{3} + \tfrac{8x-6y-2z}{3}, \tfrac{1-x+2y+2z}{3} - \tfrac{4x-3y-z}{3}, \tfrac{1+8x-4y-z}{3} - \tfrac{4x-3y-z}{3}] \\ &= [\tfrac{1-4x+5y+2z+8x-6y-2z}{3}, \tfrac{1-x+2y+2z-4x+3y+z}{3}, \tfrac{1+8x-4y-z-4x+3y+z}{3}] \\ &= [\tfrac{1+4x-y}{3}, \tfrac{1-5x+5y+3z}{3}, \tfrac{1+4x-y}{3}].\end{aligned}$$

At time t_5, agent 2 will end up playing paper and scissors with equal probability. Since agent 2 will quickly favor paper, agent 1 will have an arg max shift and favor scissors. Another arg max shift will occur after the agents' policies move a δt value equal to

$$\begin{aligned}\delta(t_6 - t_5) &= \frac{2(\frac{1+10x-7y-3z}{3} - \frac{1-8x+5y+3z}{3})}{3} \\ &= \frac{\frac{2+20x-14y-6z}{3} - \frac{2-16x+10y+6z}{3}}{3} \\ &= \frac{2+20x-14y-6z-2+16x-10y-6z}{9} \\ &= \frac{36x-24y-12z}{9} \\ &= \frac{12x-8y-4z}{3}.\end{aligned}$$

After being updated by $\delta(t_6 - t_5)$, agent 1's policy is

$$\begin{aligned}\pi_1(t_6) &= [\tfrac{1-2x+2y}{3} - \tfrac{6x-4y-2z}{3}, \tfrac{1+10x-7y-3z}{3} - \tfrac{6x-4y-2z}{3}, \tfrac{1-8x+5y+3z}{3} + \tfrac{12x-8y-4z}{3}] \\ &= [\tfrac{1-2x+2y-6x+4y+2z}{3}, \tfrac{1+10x-7y-3z-6x+4y+2z}{3}, \tfrac{1-8x+5y+3z+12x-8y-4z}{3}] \\ &= [\tfrac{1-8x+6y+2z}{3}, \tfrac{1+4x-3y-z}{3}, \tfrac{1-4x-3y-z}{3}],\end{aligned}$$

and agent 2's policy is,

$$\begin{aligned}\pi_2(t_6) &= [\tfrac{1+4x-y}{3} + \tfrac{12x-8y-4z}{3}, \tfrac{1-5x+5y+3z}{3} - \tfrac{6x-4y-2z}{3}, \tfrac{1+4x-y}{3} - \tfrac{6x-4y-2z}{3}] \\ &= [\tfrac{1+4x-y+12x-8y-4z}{3}, \tfrac{1-5x+5y+3z-6x+4y+2z}{3}, \tfrac{1+4x-y-6x+4y+2z}{3}] \\ &= [\tfrac{1+16x-9y-4z}{3}, \tfrac{1-11x+9y+5z}{3}, \tfrac{1-2x+3y+2z}{3}].\end{aligned}$$

At time t_6, agent 1 will end up playing scissors and rock with equal probability. Agent 2's policy will be favoring paper.

3.3.4 Visualizing a cycle

This completes one full cycle of the relative orderings of actions in both agents' policies. A cycle is useful in analyzing the behavior of the agents by serving as reference points. If the

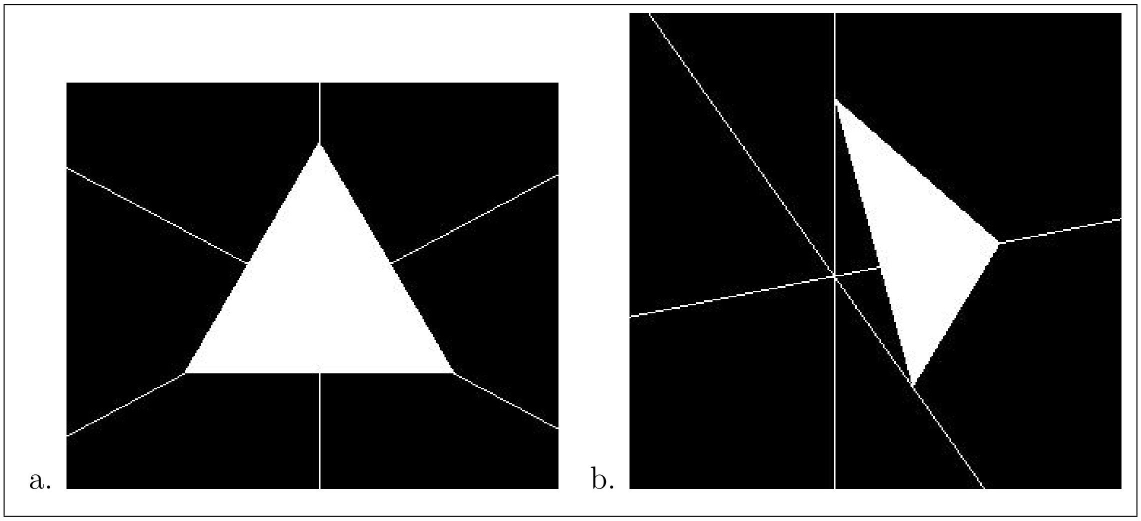

Figure 3.5: (a) Front view for the three action probability simplex. (b) Another view of Figure 3.5a.

policies are closer to the Nash equilibrium after one complete cycle, the agents' policies would be converging. If the policies have moved away from the Nash equilibrium after one complete cycle, the agents' policies would be diverging. To aid in seeing the trend, we will use a visualization of the probability hypersimplex.

Figures 3.5a and 3.5b show the probability simplex for a three action game. It is the plane $a + b + c = 1$ bounded by $a \geq 0$, $b \geq 0$, and $c \geq 0$. The pure strategies correspond to the corners of the triangle.

Figure 3.6 shows the trajectories of the agents' policies plotted on the probability simplex. The initial policies used in Figure 3.6 are given in Equations 3.4 and 3.5.

$$\pi_1 = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}], \tag{3.4}$$

$$\pi_2 = [\frac{1}{3} + .02, \frac{1}{3}, \frac{1}{3} - .02]. \tag{3.5}$$

Notice the triangular trajectories of the policies, and how the agents take turns changing the direction their policy is heading. When an agent changes the direction its policy is heading, it corresponds to an arg max shift caused by the opponent changing which action is most probable in its policy.

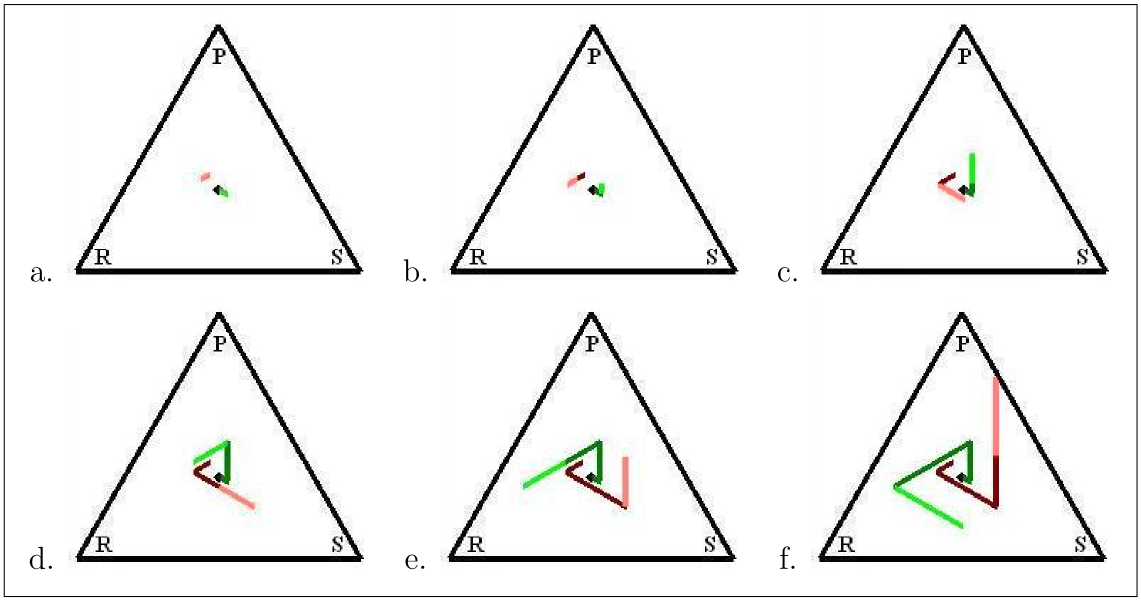

Figure 3.6: Probability simplex with policies and histories. The initial policies are given in Equations 3.4 and 3.5. Figures (a) through (f) correspond to times t_1 to t_6.

3.3.5 A useful measurement

Since a full cycle has been traversed, it would be useful to be able to say exactly how far the policy has moved from the Nash equilibrium. An obvious metric would be to take the Euclidean distance or Manhattan distance from the Nash equilibrium. Another would be to take the action with the probability that is furthest away from the Nash equilibrium and only consider that distance. These are the standard statistical metrics corresponding to the L_2-norm, the L_1-norm, and the L_∞-norm, respectively. Each of these metrics oscillate as PHC follows its triangular path.

It is desirable to have a measure that will not oscillate as PHC's triangular path is followed. Since the policies can only move in three directions and form triangular paths, the measure should have contours parallel to the movable directions and form equilateral triangles on the probability simplex. It is also useful to have a minimum value of zero occur at the Nash equilibrium, as this will make comparisons with the statistical metrics easier. With the contours of the measure defined, all that is left is to find the equation for this measure. We will construct such a measure and call it the *triangle measure*.

Let us assume that an agent's policy starts at [0,.5,.5] and travels to [1,0,0]. The line

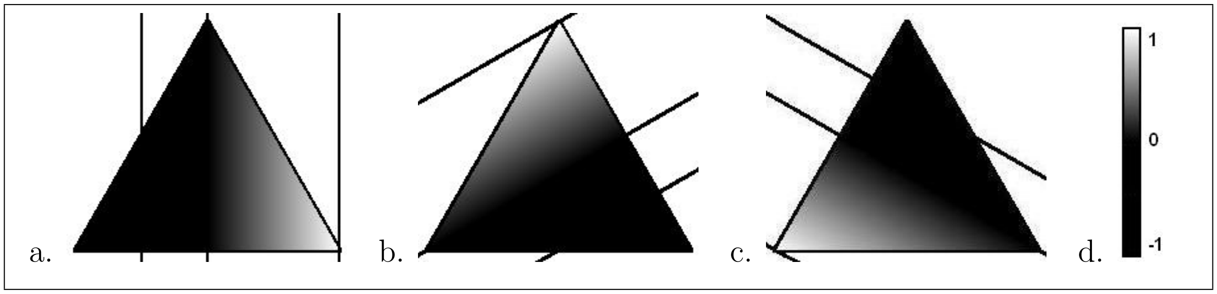

Figure 3.7: (a) c_p. (b) c_r. (c) c_s. Each graph has lines showing where the value is equal to 1, 0, and $-\frac{1}{2}$. (d) Shows the values corresponding to the graylevels. Negative values of c_p, c_r, and c_s are shown in black because they are not used to determine the final value of the triangle measure.

connecting these points is defined by $\pi(rock) = \pi(scissors)$, with the value for paper being $1 - (\pi(rock) + \pi(scissors))$. Now let us look at a parallel line. If the agent starts at [0,.4,.6] and travels to [.8,0,.2], then the agent will have traveled along the line given by $\pi(rock) + .2 = \pi(scissors)$. This may be represented equivalently by $.2 = \pi(scissors) - \pi(rock)$.

All lines following this orientation (adding to paper while removing equal amounts from rock and scissors) can be represented by $c_p = \pi(scissors) - \pi(rock)$. Notice that the closer the line is to the center of the simplex, the closer to zero the value for c_p will be. Notice also that once policies have started cycling they only move toward paper after playing scissors most often. This is due to the opponent learning to play rock against scissors, and then the agent learns to play paper against the opponent's new policy. Similar reasoning may be used to get the equation for contour lines of the other two directions, and they are $c_s = \pi(rock) - \pi(paper)$ and $c_r = \pi(paper) - \pi(scissors)$.

Figure 3.7 shows the value for each c variable when plotted on the probability simplex. Lines are added showing when the variables are equal to zero, one, and negative one half. Notice that the line corresponding to zero goes through the center of the simplex, and the line corresponding to one only touches one point on the simplex.

The measure to be used will find the c values for each action and use the largest one.

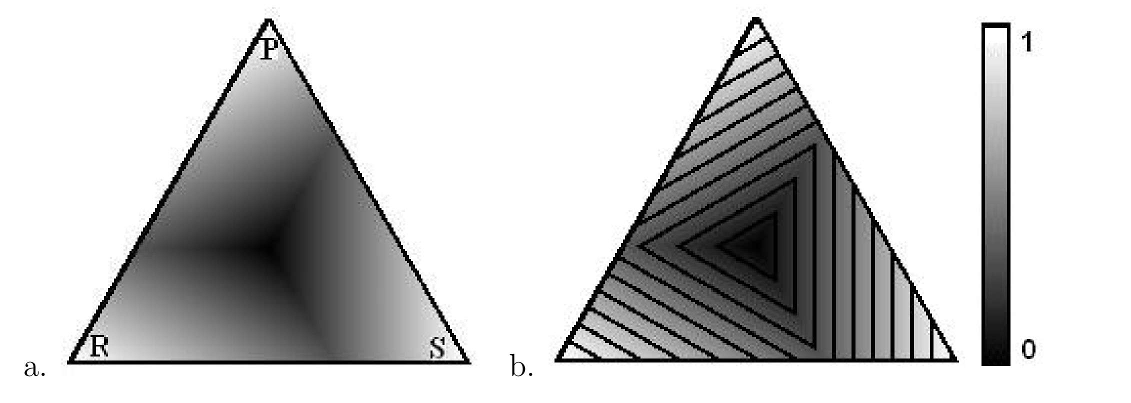

Figure 3.8: (a) Triangle measure plotted on the probability simplex. (b) 3.8a with contours marked in black.

The equation for this measure is therefore

$$\max(c_p, c_r, c_s) = \max(\pi(scissors) - \pi(rock), \pi(rock) - \pi(paper), \pi(paper) - \pi(scissors)).$$

Notice that the value of this equation is proportional to the distance for the closest approach to $\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$ of a triangle oriented as given and centered at $\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$ which goes through the specified policy.

Figures 3.8a,b show the value of the triangle measure on the probability simplex. White is shown when the value is close to one, and black is shown when the value is close to 0. Figure 3.8b has a few contours marked as well.

To verify that this equation has the contours specified, assume that the agent's policy goes through the cycle

$$\cdots \rightarrow \left[\frac{1}{3} + \epsilon, \frac{1}{3} - \epsilon, \frac{1}{3}\right] \rightarrow \left[\frac{1}{3}, \frac{1}{3} + \epsilon, \frac{1}{3} - \epsilon\right] \rightarrow \left[\frac{1}{3} - \epsilon, \frac{1}{3}, \frac{1}{3} + \epsilon\right] \rightarrow \cdots,$$

where $0 \leq \epsilon \leq \frac{1}{3}$ to avoid boundary conditions. It is easily seen that this path forms an equilateral triangle centered at $\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$ with the sides oriented to the directions that PHC can move policies.

Since the equations for the c values are linear, the values will change linearly as the policy moves from one given strategy to another. Knowing this we can check the value for the c variables at the vertices of the triangle that the agent's policy will follow, and interpolate between for the values as the policy is moving. Figure 3.9 illustrates the c

	$\left[\frac{1}{3}+\epsilon, \frac{1}{3}-\epsilon, \frac{1}{3}\right]$	$\left[\frac{1}{3}, \frac{1}{3}+\epsilon, \frac{1}{3}-\epsilon\right]$	$\left[\frac{1}{3}-\epsilon, \frac{1}{3}, \frac{1}{3}+\epsilon\right]$
c_p	ϵ	-2ϵ	ϵ
c_r	ϵ	ϵ	-2ϵ
c_s	-2ϵ	ϵ	ϵ

Figure 3.9: Chart for triangle measure example.

values for the endpoints of the given cycle. Notice that for any edge there is a c value that is constant at ϵ. Also notice that the largest value for any of the c values is ϵ. Thus, the triangle measure's value over the whole cycle is constant at ϵ.

Since ϵ can be chosen arbitrarily we see that any equilateral triangle centered at $\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$ and oriented correctly will have a constant value for the triangle measure over the entire triangle. We also note that the triangle measure has a value of 0 at $\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$, and a value of 1 at the pure strategies.

3.3.6 Analysis

In this section, we will use the triangle measure to analyze how far from the Nash Equilibrium the agents' policies have moved. Then a theorem will be given which states that PHC will diverge under any initial conditions.

Since $x + y + z = 0$, $y + z = -x$. The probability that agent 2 chooses paper at time t_6 is derived in Section 3.3.3 and repeated here.

$$\frac{1 + 16x - 9y - 4z}{3} = \frac{1 + 16x - 5y - 4(y + z)}{3} = \frac{1 + 16x - 5y + 4x}{3} = \frac{1 + 20x - 5y}{3}$$

Since y can be nearly as great as x $(z \approx -2x)$ or as small as almost $-2x$ $(z \approx x)$, after one complete cycle (paper is again at its peak percentage for agent 2) agent 2's probability of playing paper is somewhere between 5 $\left(= \frac{20x - 5(x)}{3x}\right)$ and 10 $\left(= \frac{20x - 5(-2x)}{3x}\right)$ times further away from the Nash equilibrium. Agent 1 is also much further away from playing the Nash equilibrium.

Agent 2 starts with a triangle measure of either $z-y$ or $x-z$ depending on the initial parameters chosen; thus, the maximum of these two values is the triangle measure. After one complete cycle, the triangle measure for agent 2 is

$$\begin{aligned}\pi_2(paper)-\pi_2(scissors) &= \tfrac{1+16x-9y-4z}{3}-\tfrac{1-2x+3y+2z}{3}\\ &= \tfrac{1+16x-9y-4z-1+2x-3y-2z}{3}\\ &= \tfrac{18x-12y-6z}{3}\\ &= \tfrac{18x-6y-6(y+z)}{3}\\ &= \tfrac{18x-6y+6x}{3}\\ &= \tfrac{24x-6y}{3}\\ &= 8x-2y.\end{aligned}$$

Since $\max(z-y, x-z)$ determines the initial triangle measure for agent 2, the triangle measure can range from x (at point $(x,-x,0)$) to $\approx 3x$ (near points $(x,x,-2x)$ or $(x,-2x,x)$). After one complete cycle, the value for the measure would be in the range of $6x$ ($y \approx x$) to $12x$ ($y \approx -2x$). Thus, agent 2's probabilities are diverging from the Nash equilibrium.

No matter how small x is chosen to be, after each cycle the policies for both agents will have a triangle measure value more than double what it was before the cycle. Interestingly, simulations have shown that as time approaches infinity the distance from the Nash equilibrium and therefore the amount the policies will move, increases by a factor approximately equal to Narayana's constant [1] from one time step (half of a phase) to the next. Narayana's constant is approximately 1.46557, and has a closed form of

$$\frac{1}{3}+\frac{\sqrt[3]{29+3\sqrt{93}}-\sqrt[3]{-29+3\sqrt{93}}}{3\sqrt[3]{2}}.$$

This is calculated as the only real-valued root of the function $x^3 = x^2+1$. After an entire cycle, the policies will have moved away from the equilibrium by a factor of about $1.46557^6 \approx 9.91$. We leave it to future work to uncover the underlying stucture of PHC that causes this growth constant.

3.3.7 Independence of Initial Policies

We will now show that PHC will diverge independent of initial policies. To show this we will first define a couple terms.

A Voronoi cell is the set of all points closer to a given lattice point than any other [51]. For our purposes, the lattice points will be the pure strategies in probability space. We define the *paper cell* to be the set of policies that play paper most often. Similarly for the *rock cell* and *scissors cell.* Figure 3.10a shows an arbitrary Voronoi diagram for a number of planar points [51]. Figure 3.10b shows the Voronoi diagram on the three action probability simplex. Figures 3.10c and d illustrate the Voronoi diagram for the four action probability hypersimplex.

We say that one cell *wins* against another if the pure strategy corresponding to it's lattice point gets the reward of 1 when playing against the pure strategy corresponding to the other cell's lattice point. A similar definition applies for the *losing cell.* We will also call an agent the *winning agent* if the cell containing their policy wins against the cell containing the opponent's policy. A similar definition applies for the *losing agent.*

We will use a similar definition of phase as was used previously. We relax the condition that a phase must start with an arg max shift. This allows the arbitrary starting point to begin a phase. As before, a phase ends after two arg max shifts occur. We now look at what happens after one phase.

Lemma 1. *If two PHC agents playing Shapley's game have policies in different Voronoi cells and $1 \gg \alpha \gg \delta$, then after one phase the losing agent's triangle measure will not have decreased. In other words, the losing agent will not have gotten closer to the Nash equilibrium.*

Proof. Since the agent's policies are in different Voronoi cells, one must be "winning" simply due to the nature of Shapley's game. The winning agent will move its policy further away from the equillibrium because it is winning. The losing agent will move its policy towards the cell that wins against the winning opponent's cell. In the case of both agents having a unique action with the maximum probability, this cell will be the one in

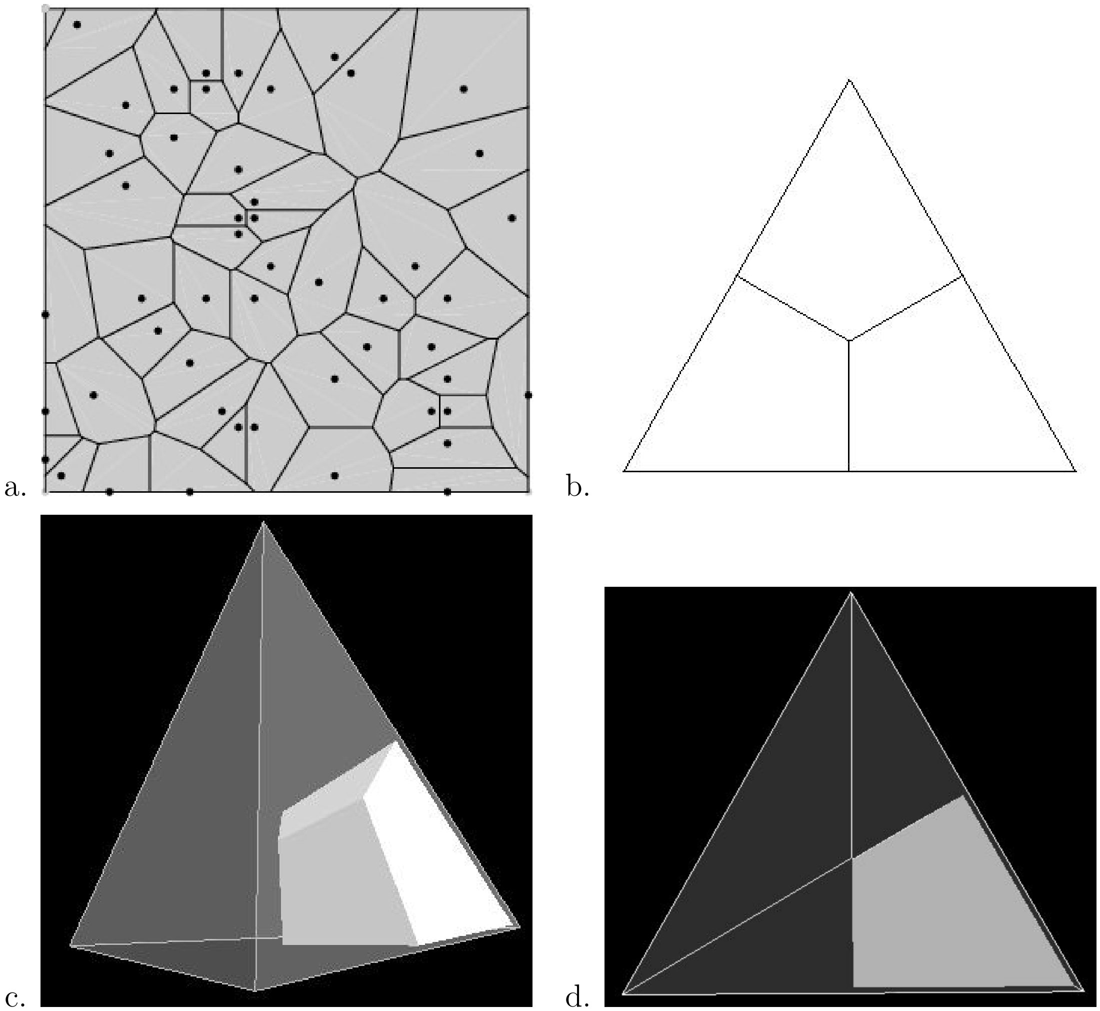

Figure 3.10: (a) Example Voronoi cells for planar points [51]. (b) Voronoi cells for the three action probability simplex. (c) One Voronoi cell (white) in the four action probability hypersimplex. (d) Bottom-up view of Figure 3.10c.

which neither agent currently "resides." For ease of computation, we will assume boundary conditions will not come into play.

Without loss of generality, let agent 1's policy be

$$\pi_1(t_0) = [3x, 3y, 3z],$$

and let agent 2's policy be

$$\pi_2(t_0) = [3a, 3b, 3c],$$

where $x \geq y$, $x \geq z$, $3x + 3y + 3z = 1$, $b > a$, $b > c$, and $3a + 3b + 3c = 1$. Notice that agent 1 (the winning player), can have two or more actions with equal probability. This means that this setup can correspond to when an arg max shift occurs.

Given that $b > a$, $b > c$, $x \geq y$, and $x \geq z$, Agent 2 is playing the rock equivalent in Shapley's game most often and agent 1 is playing the paper equivalent most often. Since agent 2 is losing, agent 2 will cause the next arg max shift. The amount the policies will need to move before this happens is $\frac{2(3b-3c)}{3} = 2b - 2c$.

The policy for agent 1 will move to

$$\pi_1(t_1) = [3x + 2b - 2c, 3y + c - b, 3z + c - b],$$

and the policy for agent 2 will be

$$\pi_2(t_1) = [3a + c - b, 2b + c, 2b + c].$$

Agent 1 will cause the next arg max shift, and the amount the policies will move is $\frac{2(3x+2b-2c-(3y+c-b))}{3} = \frac{2(3x+3b-3y-3c)}{3} = 2x + 2b - 2y - 2c$.

Agent 1's policy ends up being

$$\pi_1(t_2) = [2x + b - c + y, 2x + y - c + b, 3z - x + y + 2c - 2b].$$

Agent 2's policy ends up being

$$\pi_2(t_2) = [3a - x + y - 2b + 2c, b + 2c - x + y, 4b - c + 2x - 2y].$$

Agent 2 begins with a triangle measure of

$$\max(3b - 3a, 3a - 3c, 3c - 3b).$$

Since $b > c$, $3c - 3b$ is negative and cannot be the maximum value. This simplifies the triangle metric to

$$\max(3b - 3a, 3a - 3c).$$

The triangle metric after one phase is

$$\max(3b - 3a, 3a - 3x + 3y + 3c - 6b, 3b - 3c + 3x - 3y).$$

Notice that since $3x \geq 3y$, $b > a$, and $b > c$, the middle value is negative. So we must show that

$$\max(3b - 3a, 3a - 3c) \leq \max(3b - 3a, 3b - 3c + 3x - 3y).$$

Notice that $3b - 3a$ is a term in both maximizations. This implies that if $3b - 3a$ is greater than $3a - 3c$, then the metric after the two time steps will be at least at large as before. Assuming $3a - 3c$ is greater than $3b - 3a$ leads us to the ordering $b > a \geq c$. This ordering implies that $3b - 3c$ is greater than $3a - 3c$. Since $3x - 3y$ is non-negative, adding this can only increase the measure obtained after the two time steps.

Therefore $\max(3b - 3a, 3a - 3c) \leq \max(3b - 3a, 3b - 3c + 3x - 3y)$.

It is possible that randomness will cause the agents to vary, but with the constraint $1 \gg \alpha \gg \delta$ the effect of randomness can be neglected on average.

□

The next two lemmas follow directly from the calculations made in the previous lemma.

Lemma 2. *If two PHC agents playing Shapley's game have policies in different Voronoi cells and $1 \gg \alpha \gg \delta$, then after one "phase" the same agent will be winning and the agents' policies will still be in different Voronoi cells.*

Proof. This comes directly from the calculations from Lemma 1. Notice that after one phase, agent 2 is playing scissors most often and agent 1 just entered the rock Voronoi cell. Thus, agent 2 will be losing again after one phase, and the agents are in different Voronoi cells. Since the policies were chosen arbitrarily, this result works in general.

□

Lemma 3. *If two PHC agents playing Shapley's game have policies in different Voronoi cells and $1 \gg \alpha \gg \delta$, then after one half of a "phase" the winning agent will be losing. Also, the agent's policies will be such that lemma 1 can be used for the formerly winning agent.*

Proof. From Lemma 1, the policy for agent 1 (the winning agent) after one time step will be at

$$\pi_1(t_1) = [3x + 2b - 2c, 3y + c - b, 3z + c - b],$$

and the policy for agent 2 will be

$$\pi_2(t_1) = [3a + c - b, 2b + c, 2b + c].$$

After one time step (one half of a phase), agent 2 will have just moved into the scissors Voronoi cell (on the border) and agent 1 will still be in the paper cell. This means that agent 1 is losing and agent 2 is winning. Agent 1's policy has played paper with a frequency strictly greater than the other two actions, and Agent 2's policy has scissors tied with rock in frequency of play. Since Lemma 1 allows the winning agent to have actions with equal probabilities, the policies are such that Lemma 1 can be applied with the agents switched. Since the policies were chosen arbitrarily, save the different Voronoi cell constraint, this result holds in general.

□

Lemma 4. *If two PHC agents playing Shapley's game have policies in the same Voronoi cell and $1 \gg \alpha \gg \delta$, then their policies will eventually end up in different Voronoi cells.*

Proof. Assume the agents' policies are different and have a different distance away from causing an arg max shift. When the arg max shift occurs, one agent will stay in the same Voronoi cell while the other will change to a different cell. The agents' policies will be in different Voronoi cells and lemma 1 can be used.

Now assume the agents have different policies, but are the same distance from causing an arg max shift. The agents will both cause an arg max shift at the same time. Since the agents' policies are different but were in the same Voronoi cell, one must then be closer to causing the next arg max shift. This situation was discussed in the previous paragraph.

Lastly, assume the agents' policies are identical. They will first move their policies to cause an arg max shift for each other, and then both move towards the equilibrium. Once there, we must rely on the randomness that has been abstracted away to cause the agents to eventually move into different cells.

□

Combining these ideas leads to the following.

Theorem 1. *If $\alpha \gg \delta$ and both are decayed "fast but not too fast," then PHC will not converge on average to the Nash equilibrium when playing Shapley's game in self-play.*

Proof. Since α is being decayed, at some point it will become very small compared to 1. This allows us to use the results of Lemmas 1-4.

Lemma 4 tells us that the agents will eventually end up in different cells. Lemma 2 tells us that the expected locations of the agents will stay in different Voronoi cells from that time forward. Assume without loss of generality that agent 1 is the losing agent. Lemma 2 also says that after one phase, agent 1 will still be losing. Lemma 1 says that the agent will also not be closer to the Nash equilibrium. Therefore, agent 1's expected policies will not converge to playing its part of the Nash equilibrium.

Lemma 3 tells us that agent 2 will be losing at alternating arg max shifts. Combining this with Lemmas 1 and 2, we see that agent 2's policy will also not converge to the Nash equilibrium.

□

3.4 Empirical Results

We have shown that the expected policies of PHC will theoretically diverge when playing Shapley's game in self-play. The proof used a number of simplifying assumptions that abstract away some aspects of the PHC algorithm. Among the aspects abstracted away are learning rates, decay equations for those learning rates, and the exploration strategy. We will now discuss our approach to these aspects, and then give empirical results to support the theoretical results given.

3.4.1 Variables and Equations

PHC and WoLF-PHC are fairly sensitive to both the learning rate values used as well as the equations used to decay them over time. A number of equations have been used to decay learning rates. The following equations, used in the analysis, are a general form of decay rate equations that parameterize a large class of decay formula:

$$\alpha(t) = \frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+t}, \quad \delta_w(t) = \frac{\delta(0)*\delta_{\text{offset}}}{\delta_{\text{offset}}+t}, \quad \delta_l(t) = c\delta_w(t) \text{ for some } c \in \Re,$$

where $\alpha(0)$ and $\delta(0)$ are the initial values, and α_{offset} and δ_{offset} determine how slowly the learning rates decrease.

Bowling and Veloso [9] used the following equations for α and δ in running WoLF-PHC on three player matching pennies.

$$\alpha(t) = \frac{1}{10+\frac{t}{10000}}, \quad \delta_w(t) = \frac{1}{100+t}, \quad \delta_l(t) = 3\delta_w(t).$$

The special cases of the set of decay rate equations that will be used for this thesis include the equations used by Bowling and Veloso in [9]. When $\alpha(0) = .1$ and $\alpha_{\text{offset}} = 100000$, the α equation is equivalent to the one used in [9] for decaying α, and when $\delta(0) = .01$ and $\delta_{\text{offset}} = 100$ the δ equation is the same as used for δ.

C. Watkins proved that Q-learning would converge under certain conditions [49]. One of these conditions involves the speed at which the learning rates decay. Appendix A demonstrates the learning rate equations given follow the condition specified by Watkins.

3.4.2 Parameter Value Observations

This section will analyze how parameters affect convergence. This will be done by varying parameters for WoLF-PHC agents in self-play playing Jordan's three-person matching pennies (see Figure 3.11). The payoffs for Jordan's three-person matching pennies are

$$\left[\begin{array}{ccc} & H & T \\ H & (1,1,-1) & (-1,-1,-1) \\ T & (-1,1,1) & (1,-1,1) \end{array}\right] \qquad \left[\begin{array}{ccc} & H & T \\ H & (1,-1,1) & (-1,1,1) \\ T & (-1,-1,-1) & (1,1,-1) \end{array}\right]$$

Figure 3.11: Jordan's three-person matching pennies. The left matrix contains the payoffs to the agents when the third agent plays heads, and the right matrix contains the payoffs when the third agent plays tails.

given in Figure 3.11. The simulations will center around the parameter values chosen by Bowling and Veloso. Our objective in evaluating Jordan's game is to understand the effects of the learning rate parameters on what is learned. Figure 3.12 contains graphs that reveal parameter sensitivity.

The graphs in Figure 3.12 show the two learning parameters varied by factors of 2. The intensity of the square is proportional to the maximum distance away from the Nash equilibrium for any agent over the last 1% of the iterations. Light squares mean that the agents' policies moved far from playing their part of the Nash equilibrium, while dark squares show that the agents' policies stayed fairly close to playing the Nash equilibrium.

Figure 3.12 plots $\alpha(0)$ against $\delta(0)$. Figures 3.12a and b are simulations that use the same parameter values, differences are due to randomness and possibly pseudoconvergence (discussed in Section 3.4.6). Figure 3.12c is similar to a and b, but uses a few smaller values for $\alpha(0)$ than were used in a and b. Figure 3.12c is also run for 10 times longer to see if simulations begin to converge.

Notice that once you find parameters that converge consistently, increasing $\alpha(0)$ will not cause the simulation to diverge. Also notice the diagonal edge in the top left of Figures 3.12a and b. This shows that the ratio of $\alpha(0)$ to $\delta(0)$ is important with respect to convergence. In other words, doubling $\alpha(0)$ would likely allow $\delta(0)$ to be doubled without changing convergence. We will use these ideas to help choose parameter values for our simulations.

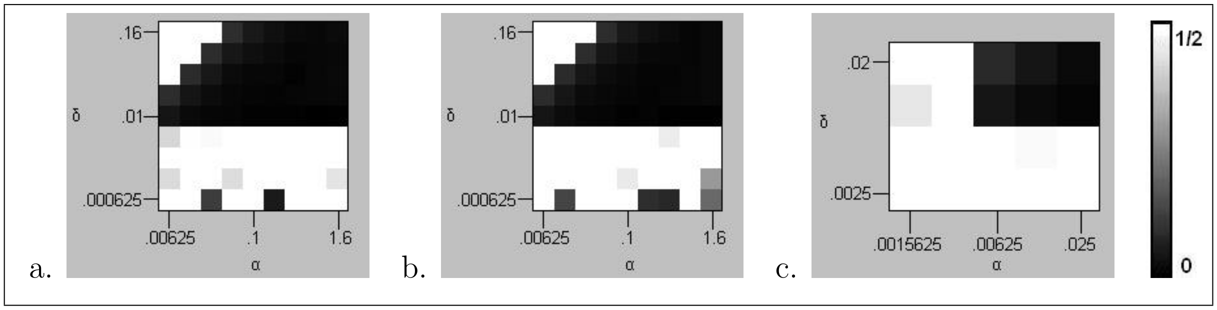

Figure 3.12: Simulations of WoLF-PHC in self-play playing Jordan's three-person matching pennies. The x-axis corresponds to the $\alpha(0)$ value, and the y-axis is the $\delta(0)$ value. The other parameters are $\alpha_{\text{offset}} = 100000$, $\delta_{\text{offset}} = 100$, $\frac{\delta_l}{\delta_w} = 3$, and $\epsilon = .1$. The simulations for (a) and (b) were run for 100 million iterations, and the simulations in (c) were run for a billion iterations. The values plotted are the largest difference from $\frac{1}{2}$ for any action probability for either agent during the last one percent of iterations. Figures 3.12a and b are obtained using different random seeds.

3.4.3 Decay Parameters and Exploration Strategies.

A subset of values for $\alpha(0)$, α_{offset}, $\delta(0)$, and δ_{offset} that will be used in simulations are given in Table 3.2. The values given in the table represent a generalization of Bowling and Veloso's choice of parameter values [9]. Parameter values not listed may be used as need warrants. For example, if an algorithm has a much larger state space than another it will need to run more iterations with slower decay rates.

The exploration strategy used can also effect learning. A few commonly used strategies are pure random, ϵ-greedy, soft-max, Boltzman, and pure exploitation. ϵ-greedy plays randomly with a certain probability (ϵ), and uses its policy the rest of the time ($1 - \epsilon$). Like in [9], we will be using ϵ-greedy for all our simulations. We will use an exploration rate of 10%, which is double the rate used in [9]. This will allow agents to react faster to changes in other agent's policies.

Using ϵ-greedy with policy hill-climbers is equivalent to changing the lower bounds on any action's probability from 0 to ϵ divided by the number of actions. This effectively forces the agent's policy to be within a smaller triangle than the regular policy simplex.

	$\alpha(0) = .1$ $\alpha_{\text{offset}} = 10^4$	$\alpha(0) = .1$ $\alpha_{\text{offset}} = 10^5$	$\alpha(0) = .01$ $\alpha_{\text{offset}} = 10^4$	$\alpha(0) = .001$ $\alpha_{\text{offset}} = 10^6$
$\delta(0) = .001$ $\delta_{\text{offset}} = 10^2$				
$\delta(0) = .001$ $\delta_{\text{offset}} = 10^3$				
$\delta(0) = .001$ $\delta_{\text{offset}} = 10^5$				
$\delta(0) = .0001$ $\delta_{\text{offset}} = 10^6$				

Table 3.2: Subset of values used in simulations

Figure 3.13 shows a mapping from the policy learned to the policy actually played. Since the opponent can only observe the policy actually played, an exploration strategy with some exploration is necessary to let policy hill-climbers play other actions once they have moved their policy to a pure strategy.

With the parameters and strategy specified, simulations can now be run. Section 3.4.4 will discuss simulations using PHC in self-play, and section 3.4.5 will discuss simulations using WoLF-PHC in self-play.

3.4.4 Empirical results for PHC and Shapley's game

Various simulations were run for PHC in self-play playing Shapley's game. A few of the results are given in Figure 3.14. Each of the experiments shown use $\epsilon = 0.1$. Other values have been empirically validated, but no substantial changes to behavior were observed.

In Figure 3.14, the x-axis corresponds to the number of learning iterations, and the y-axis is the policy at that iteration number. To reduce the number of actions plotted per graph, the policy history is only shown for one agent. This is possible due to the similarity of the agents' policies.

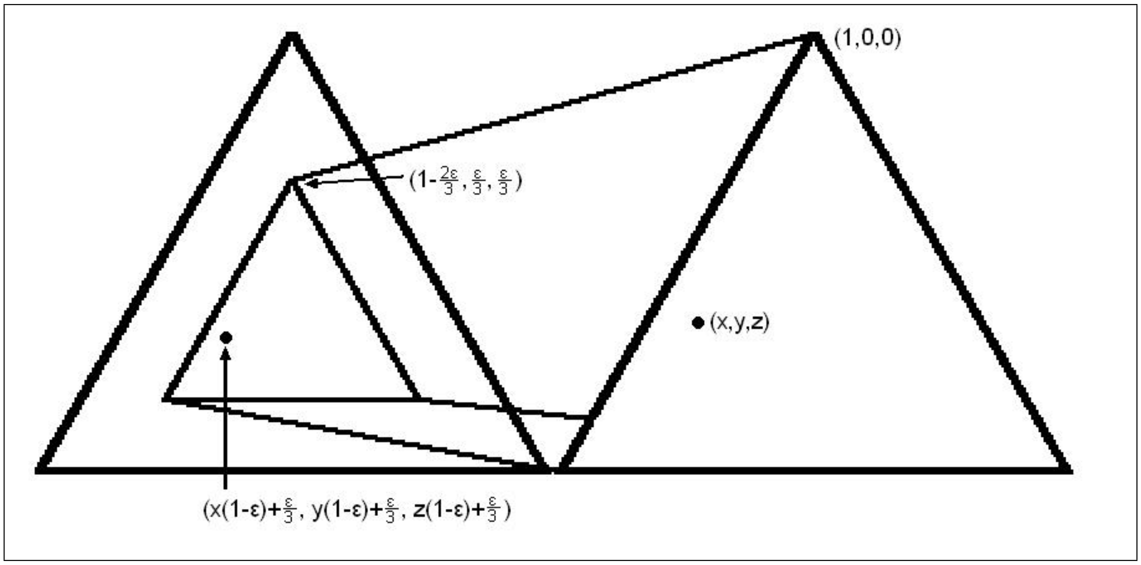
(1,0,0)
$(1-\frac{2\varepsilon}{3}, \frac{\varepsilon}{3}, \frac{\varepsilon}{3})$
(x,y,z)
$(x(1-\varepsilon)+\frac{\varepsilon}{3}, y(1-\varepsilon)+\frac{\varepsilon}{3}, z(1-\varepsilon)+\frac{\varepsilon}{3})$

Figure 3.13: The policy played compared to the policy learned when using ϵ-greedy. The triangle on the right represents the policy that is learned, the triangle on the left represents the policy actually played, and the smaller triangle within the leftmost triangle represents a mapping from the rightmost triangle to the leftmost triangle. Two points are marked to aid in illustrating this idea.

As can be seen in all the results shown, PHC diverges rather quickly. A logarithmic scale was used in half the graphs to show a more useful view of the policy's history. These graphs required a large number of iterations due to a relatively small δ_{offset}. The three graphs that use a logarithmic scale (c, d, and f) look like they start off well, but this is simply due to the logarithmic scale stretching the early iterations.

In Figure 3.14a, PHC diverges quickly and reaches a pure strategy after about 18000 iterations.

Notice the rate of change decreases around 10000 iterations. This qualitative change in learning behavior is caused by the way we handled boundary conditions. At the same time this change occurs, the policy's probability of playing rock is zero.

In Figure 3.14b, PHC again diverges quickly. A number of cycles are shown to illustrate the effect of a large δ_{offset} value. Notice that the policy's rate of change does not seem to decrease much.

Figure 3.14c is the first to use a logarithmic scale. The value for δ_{offset} is rather low,

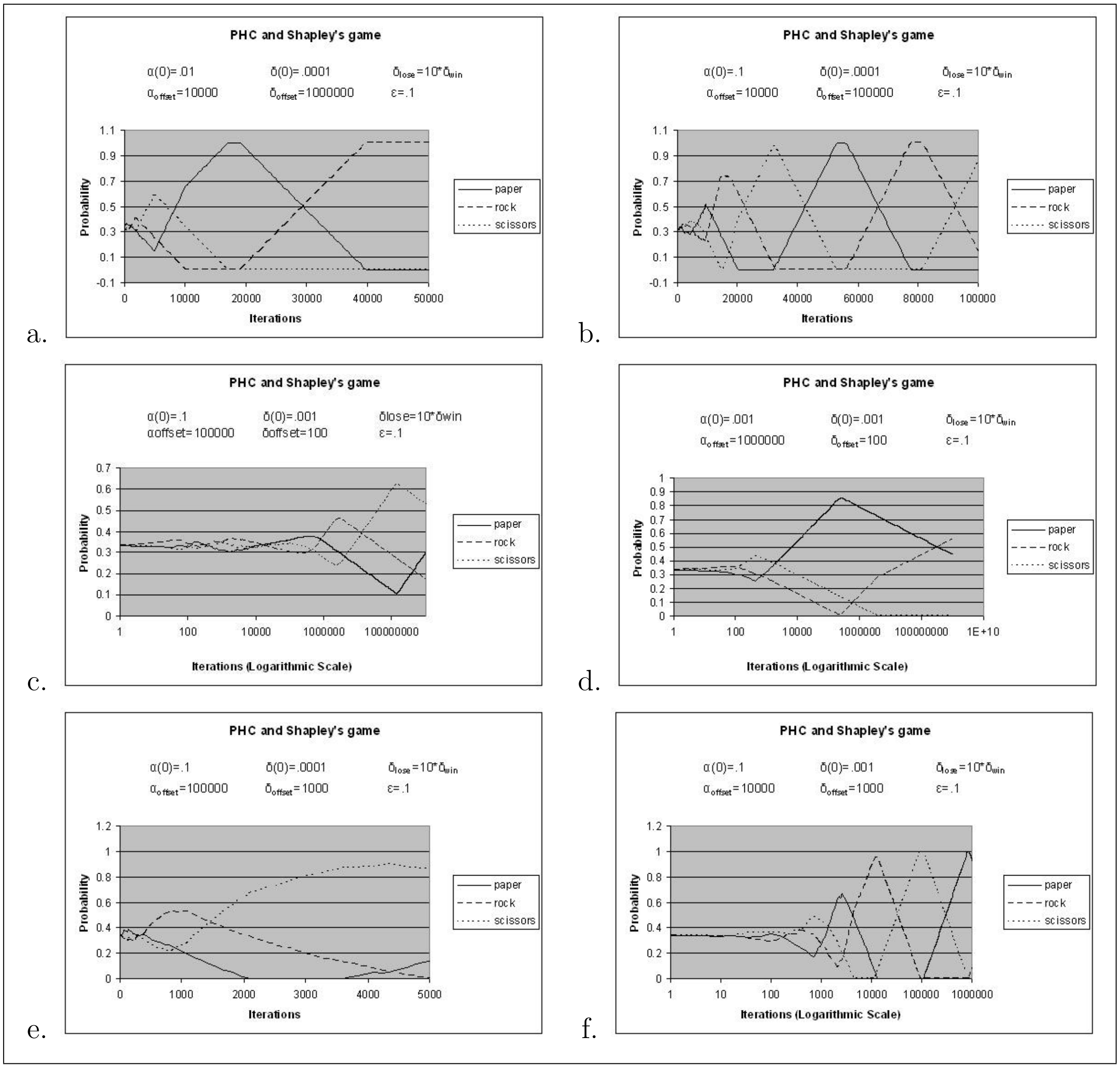

Figure 3.14: Various simulation results for PHC in self-play with Shapley's game.

so a large number of iterations are shown. Divergence is plainly seen after 10^8 iterations. Figure 3.14d also uses a logarithmic scale, but clearly diverges much quicker than Figure 3.14c.

In Figure 3.14e, divergence can be seen after around 1000 to 2000 iterations. Figure 3.14f is similar to Figure 3.14e, but a logarithmic scale is used to show more cycles. Notice that the peaks for actions' probabilities on the right side of the graph occur near powers of 10. This gives some intuition as to how quickly the algorithm is slowing down.

It is apparent that PHC has much difficulty with this game. The next section discusses how policies change when applying the WoLF principle.

3.4.5 Empirical results for WoLF-PHC and Shapley's game

The same experiments were run for WoLF-PHC as were run for PHC in the previous section. Extending PHC with the WoLF principle does help slow divergence, but does not stop it. We report results from experiments that used a value of ten for the ratio of δ_{lose} to δ_{win}; many other ratio values were tested, but no substantial differences were observed. Figure 3.15 contains a few of the more interesting results.

An interesting, but intuitive, phenomena occurs when alpha is large. When both players' policies are close to random and alpha is large, the Q-values jump around a great deal. The probability of each of the three actions increases about a third of the time, so no action changes its probability very much. At times the policy can look like it has converged, but the large α value prevents the Q-learning layer from deciding which action is best. The end result is that the empirical frequency of play is the Nash equilibrium even though the policies and Q-values are not converging. We have named this phenomena *pseudoconvergence.* Pseudoconvergence occurs in various degrees in at least 5 of the graphs in Figure 3.15.

Figure 3.15a is quite similar to Figure 3.14c. Both use a logarithmic scale. The only major difference is the parameters that caused them.

In Figure 3.15b, the agent's policy is clearly diverging after 60 million iterations. A

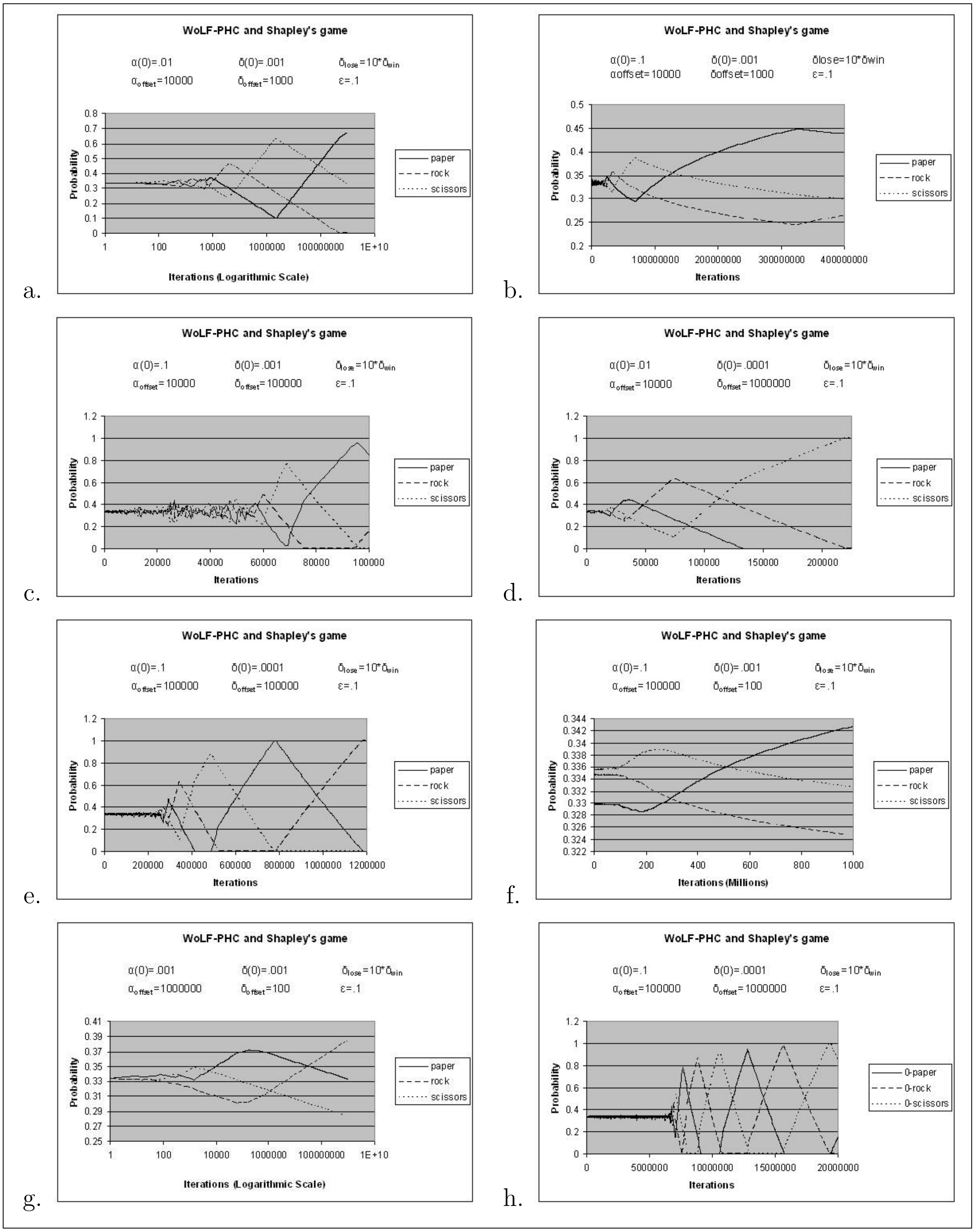

Figure 3.15: Various simulation results for WoLF-PHC in self-play with Shapley's game.

little pseudoconvergence is observed until around 20 million iterations.

Figure 3.15c shows pseudoconvergence, but in a slightly different way than the other graphs. The agent's policy is bouncing around quickly, and this is caused by large values for α_{offset} and δ_{offset}. Divergence begins by around 60 thousand iterations.

Figure 3.15d diverges quickly. Though α starts off greater than δ, the inequality flips and the policy starts adapting more quickly than the Q-values can adjust.

Figure 3.15e and Figure 3.15h both show pseudoconvergence, and both diverge quickly once α has decayed.

Figure 3.15f shows a closer look at pseudoconvergence. This graph has a small δ_{offset} value, so δ decreases rapidly. Notice the scale on the y-axis. During the first 100 million iterations, the policies seem stationary. The policy eventually diverges, but the agent's early behavior is interesting.

Figure 3.15g uses a logarithmic scale and shows cycling similar to the other graphs seen. This graph shows a very large number of iterations, but the policy is still very near the Nash equilibrium due to such a small δ_{offset} value.

At least half of the graphs in Figure 3.15 show signs of pseudoconvergence. The next section will look more closely at pseudoconvergence.

3.4.6 Pseudoconvergence

Notice the early iterations in Figure 3.15f. Though the graph only shows the policy for one agent, both agents' policies keep the same ordering (nearly the same policy) for about a hundred million iterations. It is obvious that this is not an equilibrium, though the policy stays stationary. This is what pseudoconvergence does when δ is very small.

Every simulation we ran with WoLF-PHC in self-play on Shapley's game eventually diverged. Pseudoconvergence can cause some simulations to seem to converge, but running these experiments for longer always revealed divergence.

Pseudoconvergence also occurs in a matrix game we call the *identity game*. The payoffs for this game are shown in Figure 3.16. Playing random is also a Nash equilibrium in the

$$\left[\begin{array}{cccc} & A & B & C \\ A & (1,1) & (0,0) & (0,0) \\ B & (0,0) & (1,1) & (0,0) \\ C & (0,0) & (0,0) & (1,1) \end{array}\right]$$

Figure 3.16: Payoffs for the identity game.

identity game, but policies will go towards a pareto optimal Nash equilibrium once agents start choosing the same action. Any matrix game with equivalent average rewards for each action when the opponents are playing uniformly random may show pseudoconvergence. The main problem is that with α too large, the policies cannot learn anything from the Q-values.

Figure 3.17 shows two more examples of pseudoconvergence. Notice in Figure 3.17b that α will always be equivalent to 1000δ, and yet pseudoconvergence still stops once α decreases enough. In Figure 3.17a, behavior similar to Figure 3.15f can be seen. The relative ordering of probabilities seems to stay constant, though the policies themselves are rather jagged.

3.5 Chapter Summary

In this chapter we have shown that PHC fails to converge in self-play playing Shapley's game both theoretically and empirically, and empirically shown that WoLF-PHC fails to converge as well. The fact that WoLF-PHC does converge in self-play playing the actual paper-rock-scissors game does help give a little insight into possible solutions. In the next chapter, an algorithm that does converge in self-play in Shapley's game is examined in detail.

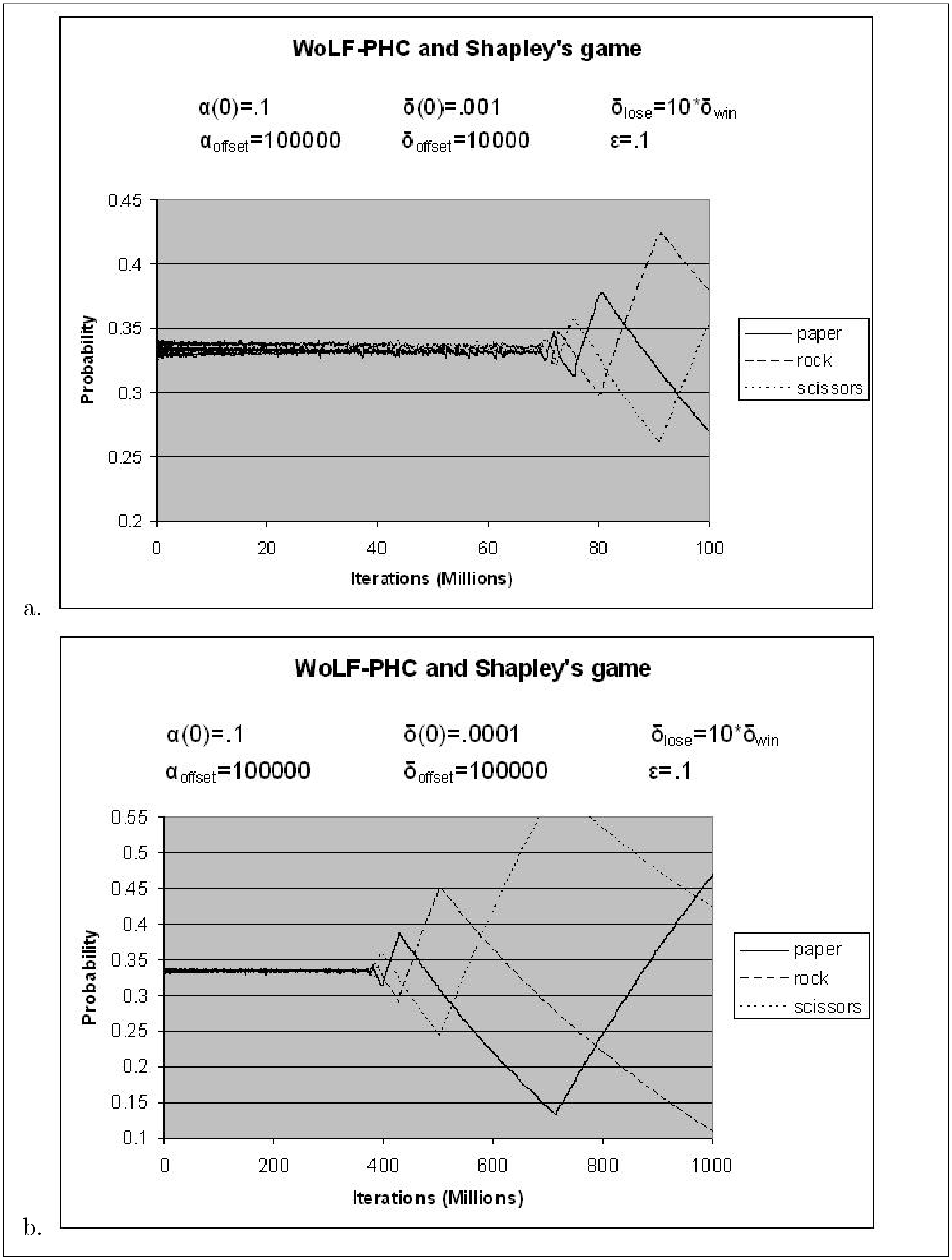

Figure 3.17: Pseudoconvergence results for WoLF-PHC in self-play with Shapley's game.

Chapter 4

Partial Commitment WoLF-PHC (PCWoLF-PHC)

In Chapter 3, PHC was shown not to converge to the Nash equilibrium when playing Shapley's game in self-play, and WoLF-PHC was empirically shown to diverge. In response to these problems, a number of modifications to WoLF-PHC were developed and tested; none of the modifications used any more information than is used by WoLF-PHC. Only one tested modification, PCWoLF-PHC, showed potential. This chapter discusses PCWoLF-PHC in detail and shows some empirical results.

Regarding the other modifications, Table 4.1, gives a summary of a few of the modifications tested and their key properties. Appendix B.1 gives details on the implementation and results of modifications that did not perform as well.

All the modifications besides PCWoLF-PHC tried to change WoLF-PHC in some way to *prevent it* from diverging in Shapley's game. By contrast, PCWoLF-PHC takes a different approach by attempting to fix WoLF-PHC *after it has started to diverge.* An overview of PCWoLF-PHC will be given followed by a step-by-step analysis and simulation results.

Name	Key property
WWoLF-PHC	Weigh the contribution of the current policy to the average policy by that of the delta value used in the current policy update.
VWoLF-PHC	Use a copy of the WoLF-PHC algorithm for each Voronoi cell in the probability simplex created by using pure strategies as the lattice points. Each copy uses the same policy.
CWoLF-PHC	Decays the reward perceived by the algorithm by a function of the number of identical outcomes in a window of time.
RPWoLF-PHC	Decay the reward perceived when a_{gfa} is played.
PWoLF-PHC	Step the policy away from a_{gfa} after the policy update steps the policy toward the action with the highest Q-value.
PCWoLF-PHC	Moves a simplex over policy space with interpolated Q-values sampled at the vertices to try and fix WoLF-PHC after it begins to cycle.

Table 4.1: WoLF-PHC modifications. a_{gfa} is defined as the member of the set of actions available to the agent, excluding the action with the highest Q-value, that is played most frequently.

4.1 Overview of PCWoLF-PHC

WoLF-PHC will converge to pure strategy Nash equilibria without much difficulty. Mixed strategy Nash equilibria pose more of a problem. There are a number of types of mixed strategy Nash equilibria. A type of mixed strategy Nash equilibrium that will be ignored in this thesis can be found in Coordination game and Battle of the Sexes.

The payoffs for Coordination game and Battle of the Sexes are listed in Figure 4.1. The mixed strategy Nash equilibrium in Coordination game occurs when both agents play C one third of the time and play D two thirds of the time. The mixed strategy Nash equilibrium in Battle of the Sexes occurs when both agents play C three quarters of the time and play D one quarter of the time. Notice that there are two pure strategy Nash equilibria in each these games, and that the mixed strategy Nash equilibria are between the pure strategy equilibria such that the actions have equal average payoffs. Notice that if an agent moves slightly towards either pure strategy equilibria, they will continue to move towards the pure strategy equilibria once the other agent changes their policy. Due to randomness, there is no possibility that WoLF-PHC could converge to those mixed strategy equilibria, but would instead converge to one of the pure strategy equilibria. Since this type of mixed strategy Nash equilibria only occurs when there are better equilibria nearby, this chapter will completely ignore them.

This leaves mixed strategy Nash equilibria that exist in games with best response loops. Whether or not WoLF-PHC will eventually converge in self-play to a given mixed strategy Nash equilibrium, the agents' policies will cycle around the equilibrium. This is because of the best response loops that give rise to these kind of equilibria, and because WoLF-PHC is a best response learner.

The idea behind PCWoLF-PHC is to restrict the algorithm's policy to a simplex that is changed based on values measured while learning. PCWoLF-PHC calculates interpolated Q-values for the mixed strategies corresponding to the vertices of the simplex. The simplex is moved based on the minimum value that the interpolated Q-values reach during a certain number of cycles of the agents' policies. This simplex represents a set of policies that

$$\left[\begin{array}{ccc} & C & D \\ C & (2,2) & (0,0) \\ D & (0,0) & (1,1) \end{array}\right] \qquad \left[\begin{array}{ccc} & C & D \\ C & (2,2) & (4,3) \\ D & (3,4) & (1,1) \end{array}\right]$$

(a) Coordination Game (b) Battle of the Sexes

Figure 4.1: Payoffs for Coordination game and Battle of the Sexes.

PCWoLF-PHC believes contain the Nash equilibrium, and the size of the simplex is a measure of how confident the algorithm is about the location of the equilibrium.

PCWoLF-PHC tries to keep a simplex roughly centered on its part of the Nash equilibrium that its policy is cycling around. This simplex is intended to allow the agent's policy to cycle around the Nash equilibrium while still being restricted. As long as the policies are still cycling, PCWoLF-PHC knows that its simplex contains the Nash equilibrium. The simplex lets the agent be less myopic than WoLF-PHC, which does not have any kind of global perspective.

Notice that if an agent's opponent is playing their part of a mixed strategy Nash equilibrium, the average reward for the first agent would be equivalent for all policies that only play actions that are included in the agent's part of the Nash equilibrium. This can be inferred by the fact that if a Nash equilibrium is being played, no agent has any incentive to change their policy. However, this does assume that there are not multiple actions that are equivalent while cycling. Any policy with an action that is not in that agent's part of the Nash equilibrium will have a smaller reward simply due to the best response dynamics needed to create the equilibrium. Playing anything other than the Nash equilibrium simply means that the agent is more exploitable, and will result in a lower reward once the other agent is able to adapt. Therefore, an agent's portion of a Nash equilibrium will result in the highest minimum average payoff when played against the kind of policies the opponent will play when policies are cycling.

Interpolating Q-values gives a mechanism to determine what average reward the agent could receive by playing a given policy. Recording the minimum of a given policy's interpolated Q-values would result in the worst average payoff the agent would expect to receive when the opposing agent plays policies similar to those played while cycling. Because the Nash equilibrium is a best response, the minimum interpolated Q-value over a number of cycles for a given policy is greater the closer the policy is to that agent's part of the Nash equilibrium. After the agents' policies have cycled, no policy would have a minimum interpolated Q-value larger than the value associated with the agent's part of the Nash equilibrium if it were not for noisy Q-values. PCWoLF-PHC requires the agents' policies to cycle a number of times; this is to try and reduce the effect of noise in Q-values.

Figure 3.4 depicts how information is passed in PHC. Figure 4.2 does the same for PCWoLF-PHC. In contrast to Figure 3.4, Figure 4.2 uses a new "Simplex" node which introduces a layer between the "World" node and the rest of the nodes. After an action is chosen by the "Simplex" node and passed to the "World" node, a reward is passed back from the "World" node. The "Simplex" node checks the reward and eventually passes the value to the "Q-learning" node. The "Q-learning" node passes the updated Q-values to the "PHC" node, which uses them to update the policy. The policy and Q-values are passed to the "Simplex" node and eventually another action is chosen as the process repeats.

Figure 4.3 shows the flow of control within the "Simplex" node for the implementation of PCWoLF-PHC. On the right side of the Figure, the "Simplex" node receives a reward from the "World" node. Before passing this value on to the "Q-learning" node, the reward is compared with the highest reward received so far. If it is higher than any reward previously seen, the "Simplex" node then resets a number of recorded values because they may not hold the right information.

Most of what the "Simplex" node does is represented on the left side of Figure 4.3. First, if it is taking too long to move the simplex, the simplex is enlarged in an attempt to promote cycling. Next, Q-values are interpolated and minimum values are recorded. Then the condition is checked to see if it is time to change the simplex, if so the simplex is moved and shrunk. Lastly, an action is chosen and passed to the "World" node.

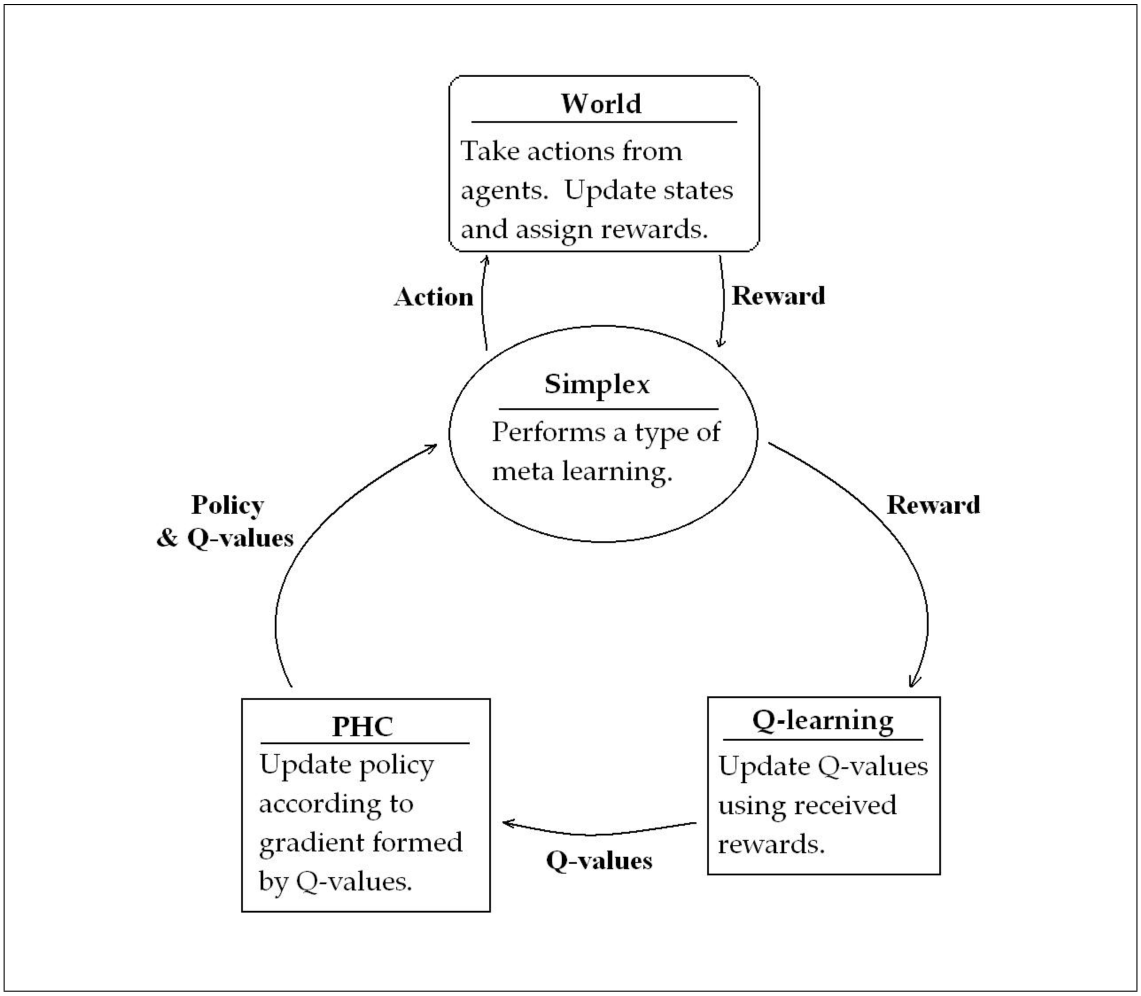
World
Take actions from agents. Update states and assign rewards.
Action
Reward
Simplex
Performs a type of meta learning.
Policy & Q-values
Reward
PHC
Update policy according to gradient formed by Q-values.
Q-values
Q-learning
Update Q-values using received rewards.

Figure 4.2: Information passing in the PCWoLF-PHC algorithm.

There are three simplices that will be referenced in this chapter. Figure 4.4 shows a graphical representation of how the simplices could look. First, the policy simplex that contains all possible probabilistic policies will be referred to by Δ_{policy}. Second, the subsimplex to which policies are restricted will be referred to by $\Delta_{\text{restricted}}$. This is the simplex that will be moved and shrunk while PCWoLF-PHC is learning. Third, a simplex will be used that has the same center of mass as $\Delta_{\text{restricted}}$, but is smaller by a factor of f_{small}. This simplex will be referred to as Δ_{shrink}.

The implementation for PCWoLF-PHC is given in Tables 4.2, 4.3, and 4.4. A list of the parameters and variables used in PCWoLF-PHC as well as descriptions for them can be found in Tables 4.5, 4.6, and 4.7.

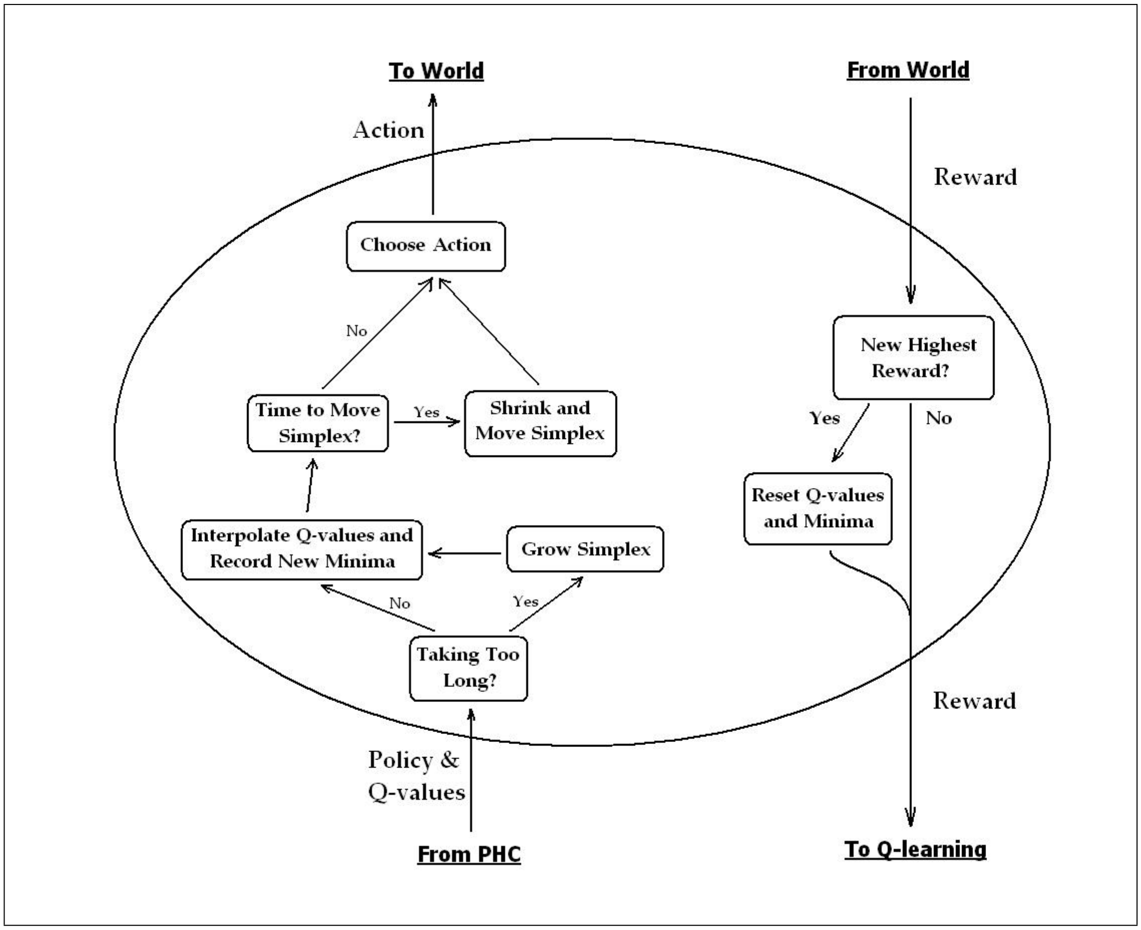

Figure 4.3: Flow of control within the "Simplex" node from Figure 4.2.

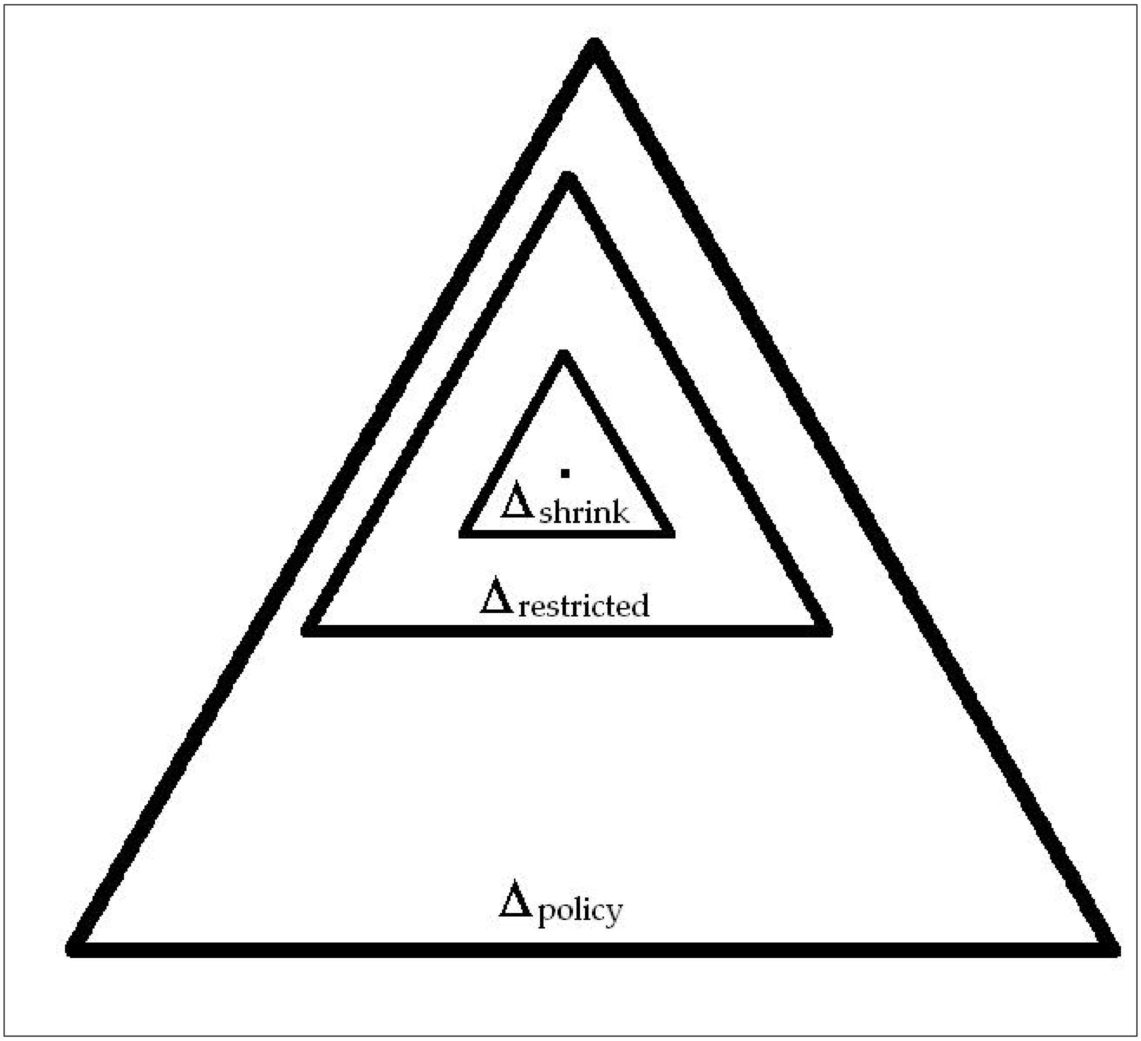

Figure 4.4: Graphical representation of a possible configuration of the three simplices that are used by PCWoLF-PHC.

```
1. Let $\alpha, \delta_l > \delta_w, \eta \in (0, \frac{1}{|A_i|}]$ and $\epsilon \in (0, 1]$ be values which are decayed over time.
   Let $\tau_{\text{best}} \in (0, 1),\ \tau_{\text{streak}} \in \mathbb{N} \geq 2,\ \tau_{\text{shrink}} \in \mathbb{N} \geq 2,\ f_{\text{stop}} \geq 2,\ f_{\text{small}} \in (0, 1),$
   $f_{\text{middle}} \in [0, 1],\ f_{\text{move}} \in (0, 1),\ f_{\text{shrink}} \in (1, \infty),\ p_{\text{stop}} \in (0, \infty), \gamma \in [0, 1)$ be constants.
   Initialize,
     $Q(s, a) \leftarrow 0,$   $\pi(s, a) \leftarrow \frac{1}{|A_i|},$   $\bar{\pi}(s, a) \leftarrow \frac{1}{|A_i|},$   $C(s) \leftarrow 0,$
     $C_\epsilon(s) \leftarrow 0,$   $C_\eta(s) \leftarrow 0,$   $r_{\text{high}} \leftarrow -\infty,$   $\delta_{\text{since}}(s) \leftarrow 0,$
     $\delta_{\text{previous}}(s) \leftarrow \infty,$   $t_{\text{shrink}}(s) \leftarrow 0,$   $a_{\text{best}}(s) \leftarrow \emptyset,$   $\delta_{\text{best}}(s) \leftarrow 0,$
     $C_{\text{streak}}(s, a) \leftarrow 0,$   $Center(s, a) \leftarrow \frac{1}{|A_i|},$   $InCycle(s) \leftarrow \emptyset,$   $a_{\text{laststreak}}(s) \leftarrow \emptyset.$
2. Repeat,
   (a) From state $s$ select action $a$ from $\pi(s, a)$ with probability $\epsilon$, and from
       $Center(s, a)$ with probability $(1 - \epsilon)$
   (b) Observing reward $r$ and next state $s'$,
       i. If $r > r_{\text{high}}$, then
          A. $\forall s'' \in S\ \forall a \in A_i \quad Q_{\text{min}}(s'', a) \leftarrow \frac{r}{1-\gamma},\ Q(s'', a) \leftarrow \frac{r}{1-\gamma}.$
          B. $\forall s'' \in S\ \forall a \in A_i \quad Q_{\text{shrink}}(s'', a) \leftarrow \frac{r}{1-\gamma},\ Q_{\text{middle}}(s'') \leftarrow \frac{r}{1-\gamma}.$
          C. $r_{\text{high}} \leftarrow r$
       ii. $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha\left(r + \gamma \max_{a'} Q(s', a')\right).$
   (c) Update estimate of average policy, $\bar{\pi}$. Same as WoLF-PHC.
   (d) Step $\pi$ closer to the optimal policy w.r.t. Q. Same as WoLF-PHC.
   (e) Decrease $C_\epsilon(s)$ if $\Delta_{\text{restricted}}$ has not moved in a while.
       Let $\delta$ be the value from the previous step.
       i. $\delta_{\text{since}} \leftarrow \delta_{\text{since}} + \delta.\ \ \delta_{\text{best}} \leftarrow \delta_{\text{best}} + \delta.$
       ii. $C_\epsilon(s) \leftarrow \begin{cases} \max(0, \frac{C_\epsilon(s) - p_{\text{stop}}}{f_{\text{shrink}}}) & \text{if } \left\lfloor \frac{(\delta_{\text{best}} - \delta) * \tau_{\text{streak}}}{\delta_{\text{previous}}} \right\rfloor < \left\lfloor \frac{\delta_{\text{best}} * \tau_{\text{streak}}}{\delta_{\text{previous}}} \right\rfloor \\ & \text{and } \left\lfloor \frac{\delta_{\text{best}} * \tau_{\text{streak}}}{\delta_{\text{previous}}} \right\rfloor \geq f_{\text{stop}} \\ C_\epsilon(s) & \text{otherwise} \end{cases}$
```

Table 4.2: PCWoLF-PHC algorithm for player i, continued in Table 4.3.

```
(f) Update $Q_{\text{min}}(s,a)$ and $Q_{\text{shrink}}(s,a)$ for all $a$, and update $Q_{\text{middle}}(s)$.
    i. $\forall a' \in A_1\ Q_{\text{min}}(s,a') \leftarrow min(Q_{\text{min}}(s,a'), \sum_{a''}(Q(s,a'') * simplex(s,a',a'')))$
       where $simplex(s,a',a'') = (1-\epsilon) * Center(s,a') + \begin{cases} \epsilon & \text{if } a' = a'' \\ 0 & \text{otherwise} \end{cases}$
    ii.$\forall a' \in A_1\ Q_{\text{shrink}}(s,a') \leftarrow min(Q_{\text{shrink}}(s,a'), \sum_{a''}(Q(s,a'') * shrunk(s,a',a'')))$
       where
       $shrunk(s,a',a'') = (1-\epsilon) * Center(s,a') + \epsilon * \begin{cases} f_{\text{small}} + \frac{1-f_{\text{small}}}{|A_i|} & \text{if } a' = a'' \\ \frac{1-f_{\text{small}}}{|A_i|} & \text{otherwise} \end{cases}$
    iii. $Q_{\text{middle}}(s) \leftarrow min(Q_{\text{middle}}(s), \sum_{a''}(Q(s,a'') * middle(s,a',a'')))$
       where $middle(s,a',a'') = (1-\epsilon) * Center(s,a') + \frac{\epsilon}{|A_i|}$
(g) Update $C_{\text{streak}}(s,a)$ if necessary.
    If $\arg\max_{a'} Q(s,a') \neq a_{\text{best}}(s)$, and $a_{\text{best}}(s) \neq a_{\text{laststreak}}$ then
    i. $a_{\text{best}}(s) \leftarrow \arg\max_{a'} Q(s,a')$
    ii. If $\delta_{\text{best}} \geq \tau_{\text{best}}$, then $C_{\text{streak}}(s, a_{\text{best}}(s)) \leftarrow C_{\text{streak}}(s, a_{\text{best}}(s)) + 1$
    iii. $\delta_{\text{best}} \leftarrow 0$
    iv. $a_{\text{laststreak}}(s) \leftarrow a_{\text{best}}(s)$
    v. $InCycle(s) \leftarrow InCycle(s) \cup \{a_{\text{best}}(s)\}$
(h) Move $\Delta_{\text{restricted}}$ if it is time to do so.
    i. If $\max_{a'} C_{\text{streak}}(s,a') > \tau_{\text{streak}}$ and $\forall a' \in InCycle(s)\ C_{\text{streak}}(s,a') > 0$,
       then $MoveSimplex(s)$
```

Table 4.3: PCWoLF-PHC algorithm for player i, continued from Table 4.2.

```
1. Reset counts for policy movement
    (a) $\forall a' \in A_1 \;\; C_{\text{streak}}(s, a') \leftarrow 0$
2. Update $t_{\text{shrink}}(s)$ and update $C_\epsilon$ if necessary
    (a) If $Q_{\text{middle}} > \max_{a'} Q_{\text{shrink}}(s, a)$
        i. $t_{\text{shrink}}(s) \leftarrow t_{\text{shrink}}(s) + 1$
        ii. $\epsilon_{\text{before}} \leftarrow \epsilon$
        iii. $C_\epsilon(s) \leftarrow \begin{cases} \max(C_\epsilon(s) * f_{\text{shrink}}, 0) & \text{if } t_{\text{shrink}}(s) \geq \tau_{\text{shrink}} \\ C_\epsilon(s) & \text{otherwise} \end{cases}$
        iv. If $t_{\text{shrink}}(s) \geq \tau_{\text{shrink}}$, then $\forall a' \, Center(s, a') \leftarrow \frac{Center(s,a')*(1-\epsilon_{\text{before}})+\frac{\epsilon_{\text{before}}-\epsilon}{numactions}}{1-\epsilon}$
        v. If $t_{\text{shrink}}(s) \geq \tau_{\text{shrink}}$, then $t_{\text{shrink}}(s) \leftarrow 0$
    (b) Otherwise, $t_{\text{shrink}} \leftarrow 0$
3. Set $\delta_{\text{previous}}$ to $\delta_{\text{since}}$, and set $\delta_{\text{since}}$ to 0
4. Increment counts. $C_\eta(s) \leftarrow C_\eta(s) + 1$, $C_\epsilon(s) \leftarrow C_\epsilon(s) + 1$
5. $\forall a' \in A_1 \;\; Q_{\min}(s, a') \leftarrow \frac{r_{\text{high}}}{1-\gamma}$
6. Move the center of $\Delta_{\text{restricted}}$
   Let $AvgQ = \frac{\sum_{a'} Q_{\min}(s,a')}{|A_1|}$, $MaxQ = \max_{a'} Q_{\min}(s, a')$, and $MinQ = \min_{a'} Q_{\min}(s, a')$
    (a) If $Q_{\text{middle}} > \max_{a'} Q_{\text{shrink}}(s, a)$, then $f_{\text{combined}} \leftarrow f_{\text{move}} * f_{\text{middle}}$
        otherwise $f_{\text{combined}} \leftarrow f_{\text{move}}$
    (b) $\forall a' \in A_1 \;\; Center(s, a') \leftarrow Center(s, a') + \frac{Q_{\min}(s,a')-AvgQ}{MaxQ-MinQ} * f_{\text{combined}} * \epsilon$
    (c) Constrain $Center(s, a)$ such that
        i. $\forall a' \in A_1 \;\; \eta \leq Center(s, a') \leq 1 - \eta$
        ii. $\sum_{a'} Center(s, a') = 1$
```

Table 4.4: *MoveSimplex*(s) method of PCWoLF-PHC algorithm for player i.

Name	Parameter or Variable?	Description
α	Parameter	Learning rate used for updating Q-values. Value is decayed over time.
δ_l	Parameter	Step size for updating policy when “losing.” Value is decayed over time.
δ_w	Parameter	Step size for updating policy when “winning.” Value is decayed over time.
γ	Parameter	Discount factor for future rewards.
$Q(s,a)$	Variable	“Quality” of choosing action a in state s.
$\pi(s,a)$	Variable	Percent of time to choose action a in state s.
$\bar{\pi}(s,a)$	Variable	Average policy.
$C(s)$	Variable	Number of times state s has been visited.
$Center(s,a)$	Variable	Policy the algorithm believes it should play.
η	Parameter	Minimum value for any $Center(s,a)$. Prevents the algorithm from playing one action 100% of the time (i.e., no exploration). Value is decayed over time.
ϵ	Parameter	Probability of choosing an action from $\pi(s,a)$. $1-\epsilon$ is the probability of choosing an action from $Center(s,a)$. Value is decayed over time.
τ_{best}	Parameter	Threshold for the amount the WoLF-PHC layer's policy needs to move before the action with the highest Q-value is incremented in $C_{\text{streak}}(s,a)$.
τ_{streak}	Parameter	Threshold for $C_{\text{streak}}(s,a)$. Once this value is reached (along with another condition), $MoveSimplex(s)$ is called.

Table 4.5: Descriptions of parameters and variables in PCWoLF-PHC. Continues in Table 4.6

Name	Parameter or Variable?	Description
τ_{shrink}	Parameter	Threshold for the number of times in a row that $Q_{\text{middle}}(s)$ has been greater than $\max_{a'} Q_{\text{shrink}}(s, a)$. Once this threshold is reached, epsilon is multiplied by f_{shrink}
f_{stop}	Parameter	If the algorithm takes more than f_{stop} times as long to move $\Delta_{\text{restricted}}$ as it did the previous time, $C_\epsilon(s)$ is decreased. Amount is based on f_{shrink} and p_{stop}.
f_{small}	Parameter	Used in equation to calculate policies corresponding to $Q_{\text{shrink}}(s, a)$.
f_{middle}	Parameter	Factor to slow movement of $\Delta_{\text{restricted}}$ if the interpolated Q-value for the center of $\Delta_{\text{restricted}}$ is greater than $\max_a Q_{\text{shrink}}(s, a)$.
f_{move}	Parameter	Amount to move $\Delta_{\text{restricted}}$ relative to size of Δ_{policy}.
f_{shrink}	Parameter	$C_\epsilon(s)$ is multiplied or divided by this depending on whether the algorithm is to shrink or grow $\Delta_{\text{restricted}}$.
p_{stop}	Parameter	Subtracted from $C_\epsilon(s)$ if the algorithm is to grow $\Delta_{\text{restricted}}$.
$Q_{\text{min}}(s, a)$	Variable	Minimum interpolated Q-values corresponding to vertices of $\Delta_{\text{restricted}}$ since the last time $\Delta_{\text{restricted}}$ was moved.
$Q_{\text{shrink}}(s, a)$	Variable	Minimum interpolated Q-values corresponding to vertices of Δ_{shrink}.
$Q_{\text{middle}}(s)$	Variable	Minimum interpolated Q-values corresponding to the center of $\Delta_{\text{restricted}}$.
$C_{\text{streak}}(s, a)$	Variable	Count for the number of times the PHC layer has moved $\pi(s)$ in one continuous direction for at least τ_{streak}.

Table 4.6: Descriptions of parameters and variables in PCWoLF-PHC. Continued from Table 4.5, and continues in Table 4.7.

Name	Parameter or Variable?	Description
$C_\epsilon(s)$	Variable	Count used to decay ϵ.
$C_\eta(s)$	Variable	Count used to decay η.
r_{high}	Variable	Highest payoff seen by the algorithm in any state.
$\delta_{\text{since}}(s)$	Variable	Amount the PHC layer has moved $\pi(s)$ since the last time $\Delta_{\text{restricted}}$ was moved.
$\delta_{\text{previous}}(s)$	Variable	Amount the PHC layer had moved $\pi(s)$ before the last time $\Delta_{\text{restricted}}$ was moved and after the previous time $\Delta_{\text{restricted}}$ was moved.
$t_{\text{shrink}}(s)$	Variable	Count for the number of times that $Q_{\text{middle}}(s)$ has been the largest minimum interpolated Q-value. Used to determine when to shrink $\Delta_{\text{restricted}}$ quickly.
$a_{\text{best}}(s)$	Variable	The action that had the largest Q-value the last time the policy was updated.
$\delta_{\text{best}}(s)$	Variable	Distance the PHC layer has moved $\pi(s)$ without and arg max shift.
$InCycle(s)$	Variable	Set of actions that the policy believes are in the cycle.
$a_{\text{laststreak}}(s)$	Variable	The action that was favored the last time that $C_{\text{streak}}(s,a)$ was updated.

Table 4.7: Descriptions of parameters and variables in PCWoLF-PHC. Continued from Table 4.6.

4.2 Details of PCWoLF-PHC

The previous section gave a brief overview of the PCWoLF-PHC algorithm. This section is intended to fill in the details and show how they fit into the big picture. This section will be organized based on Figure 4.3 with subsections referencing seperate nodes in the figure.

4.2.1 "Choose Action"

Step 2a in Table 4.2 corresponds to the "Choose Action" node of Figure 4.3. Like all reinforcement learning algorithms, PCWoLF-PHC needs to choose an action to receive a reward and continue learning. This is where the algorithm chooses what action it will take based upon a probabilistic policy.

An example to illustrate this concept is the following. Let $\pi(s) = [\frac{1}{6}, \frac{1}{3}, \frac{1}{2}]$. Choosing an action from this policy would be similar to rolling a die and choosing an action based on that. If a "1" was rolled, the first action is chosen and played. If either a "2" or a "3" is rolled, the second action is chosen and played. Otherwise the third action is taken.

The policy that PCWoLF-PHC chooses from is a weighted blend of two policies. The first policy that PCWoLF-PHC blends is $\pi(s)$. Recall that $\pi(s)$ is the policy that is hill climbing in PHC, and is always moving toward the action with the highest Q-value. The second is $Center(s)$. $Center(s)$ is essentially a guess by the "Simplex" node as to the policy it thinks is its part of a Nash equilibrium. This blending is based on how confident the "Simplex" node is of the guess, where $\epsilon = 1$ means the node has no confidence in the guess and $\epsilon = 0$ means the node has total confidence in the guess. The way the policies are blended limits the policies to be used as a basis for choice to that of a simplex centered at $Center(s) * (1 - \epsilon) + \frac{\epsilon}{|A_i|}$. This simplex is being referred to as $\Delta_{\text{restricted}}$. The exact policy being used within $\Delta_{\text{restricted}}$ will then be determined by $\pi(s)$.

A pictorial representation of the policy the algorithm would actually be playing at a given point in time can be found in Figure 4.5. Figure 4.5 uses a two-action game to simplify the representations. The policy used as a basis for choice is obtained by

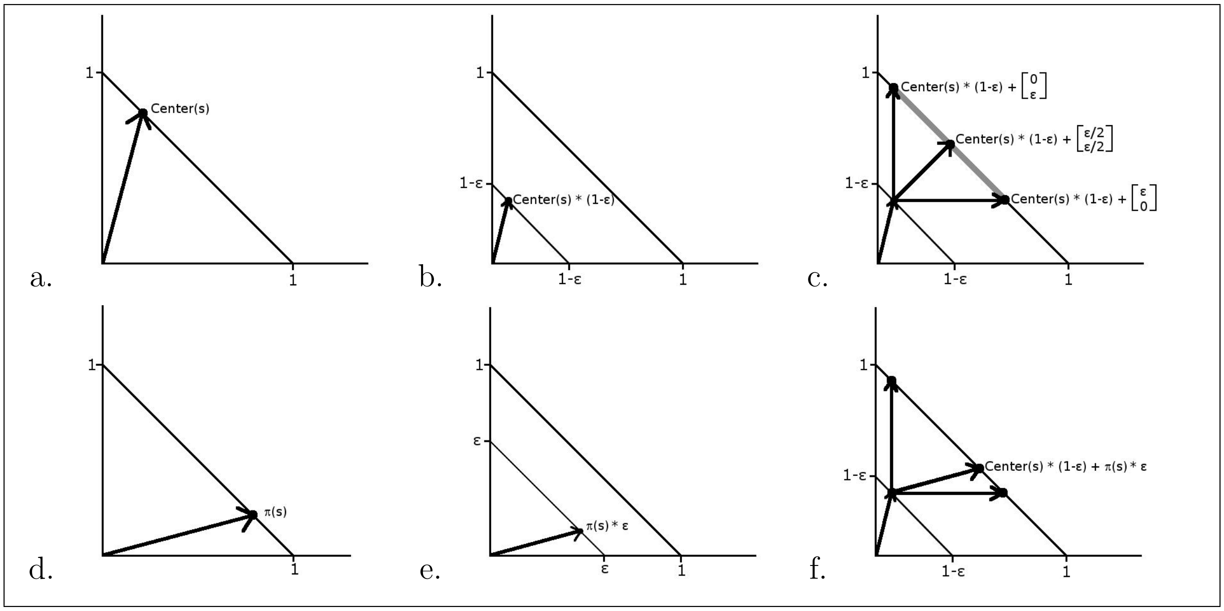
a.
1
Center(s)
1
b.
1
1-ε
Center(s) * (1-ε)
1-ε
1
c.
1
Center(s) * (1-ε) + $\begin{bmatrix}0\\ \varepsilon\end{bmatrix}$
Center(s) * (1-ε) + $\begin{bmatrix}\varepsilon/2\\ \varepsilon/2\end{bmatrix}$
1-ε
Center(s) * (1-ε) + $\begin{bmatrix}\varepsilon\\ 0\end{bmatrix}$
1-ε
1
d.
1
π(s)
1
e.
1
ε
π(s) * ε
ε
1
f.
1
1-ε
Center(s) * (1-ε) + π(s) * ε
1-ε
1

Figure 4.5: (a) Vector corresponding to $Center(s, a)$ for a two-action game. (b) $Center(s, a)$ scaled by $1 - \epsilon$. (c) Simplex to which policies are limited (in gray) with policies corresponding to vertices and the midpoint marked. (d) Vector corresponding to $\pi(s, a)$. (e) $\pi(s, a)$ scaled by ϵ. (f) Policy PCWoLF-PHC plays. This corresponds to summing the scaled vectors from b and e.

linearly interpolating between the vectors $Center(s)$ and $\pi(s)$ based on the value of ϵ. Linear interpolation is used because it is a simple and standard way of blending two values. This linear interpolation is calculated by adding scaled versions of both vectors, $(1 - \epsilon) * Center(s) + \epsilon * \pi(s)$, and is shown this way in Figures 4.5b, e, and f. Figure 4.5c highlights $\Delta_{\text{restricted}}$ in gray. Notice that $\Delta_{\text{restricted}}$ is the set of all policies that may result when changing the value of $\pi(s)$, but keeping the values of $Center(s)$ and ϵ constant.

Once an action is chosen and played, PCWoLF-PHC gets a reward from the world. The next section describes what PCWoLF-PHC does with this reward before passing it to the Q-learning layer.

4.2.2 "New Highest Reward?" and "Reset Q-values and Minima"

Step 2b,i and its substeps correspond to the "New Highest Reward?" node of Figure 4.3. PCWoLF-PHC records the minimum value that interpolated Q-values have reached during a given amount of time. The main values of this type that are used by PCWoLF-PHC are recorded in $Q_{\text{min}}(s, a)$. Because we do not wish to introduce minima that do not accurately represent the dynamics of the game, the reward received is compared with the highest seen so far (r_{high}). These artificially created minima may be introduced a number of ways including starting Q-values too low or using an α value that is far too large.

The minimum value of interpolated Q-values for policies of interest was chosen to be used because it approximates the minimum average payoff for those policies of interest against common strategies played by the opposing player. The Q-values must be started at the highest possible Q-value according to the rewards or an artificial minimum may be introduced by starting the Q-values off too low initially. The highest possible Q-value may be computed using γ and the largest reward. Since PCWoLF-PHC does not require this information beforehand, it simply resets all Q-values and minimums every time a reward is received that is larger than the largest that has been received previously.

PCWoLF-PHC uses a number of variables that store minimum interpolated Q-values. $Q(s, a)$ are the regular Q-values. $Q_{\text{min}}(s, a)$ are the minimum interpolated Q-values for the vertices of $\Delta_{\text{restricted}}$. $Q_{\text{shrink}}(s, a)$ are the minimum interpolated Q-values for the vertices of Δ_{shrink}, a smaller simplex that has the same center of mass as $\Delta_{\text{restricted}}$. $Q_{\text{middle}}(s)$ is the interpolated Q-value for the center of mass of $\Delta_{\text{restricted}}$. $Q_{\text{shrink}}(s, a)$ and $Q_{\text{middle}}(s)$ were recorded to help the algorithm determine if $\Delta_{\text{restricted}}$ is roughly centered on the Nash equilibrium or not. It tries to make this determination by checking if the minimum interpolated Q-value associated with the center of $\Delta_{\text{restricted}}$ ($Q_{\text{middle}}(s)$) is greater than any of the interpolated Q-values associated with the vertices of Δ_{shrink} ($Q_{\text{shrink}}(s, a)$). If this is the case, there is no incentive to move $\Delta_{\text{restricted}}$ very much. If there is not an incentive to change much, then the center of $\Delta_{\text{restricted}}$ likely contains this agent's part of a Nash equilibrium. The details concerning how the simplex is moved will be covered in Section 4.2.6

Step 2b,ii is the familiar Q-value update equation. Steps 2c and 2d are the exact same steps as in WoLF-PHC.

4.2.3 "Taking Too Long?" and "Grow Simplex"

Step 2e,i updates δ_{since}, which is the variable containing the total amount $\pi(s)$ has moved since $\Delta_{\text{restricted}}$ had last moved. Also updated is the variable δ_{best} which holds how far $\pi(s)$ has moved while a given action has had the highest Q-value. δ_{previous} holds the total amount $\pi(s)$ moved between the last time $\Delta_{\text{restricted}}$ was moved and the time before that.

Step 2e,ii corresponds to both the "Taking Too Long?" and the "Grow Simplex" nodes of Figure 4.3. f_{stop} can be thought of as how much longer it takes to cycle compared to before the last time $\Delta_{\text{restricted}}$ was moved. If the value of δ_{best} gets to be larger than f_{stop} times the amount $\pi(s)$ needed to move the previous time (δ_{previous}) divided by the number of cycles required before $\Delta_{\text{restricted}}$ is moved (τ_{streak}), then the policy has probably stopped cycling. If the policy has stopped cycling, then $\Delta_{\text{restricted}}$ may not allow the agent's policy to move far enough towards the next joint policy in the best response loop. To attempt to get the policy to start cycling again, $\Delta_{\text{restricted}}$ is then increased in size. C_η and C_ϵ are the count variables used to decay η and ϵ, respectively. Notice that decreasing C_ϵ results in increasing the size of $\Delta_{\text{restricted}}$.

4.2.4 "Interpolate Q-values and Record New Minima"

Step 2f in Table 4.3 corresponds to the "Interpolate Q-values and Record New Minima" node of Figure 4.3. Notice that to calculate interpolated Q-values simply requires a dot product of the current Q-values and the policy for which the Q-value should be interpolated. If the value calculated happens to be smaller than the minimum interpolated Q-value seen so far for that given policy, then the value recorded will be changed to the new value.

4.2.5 "Time to Move Simplex?"

Steps 2g and 2h correspond to the "Time to Move Simplex?" node of Figure 4.3. If δ_{best} is large enough ($\geq \tau_{\text{best}}$) when an arg max shift occurs, then a count variable ($C_{\text{streak}}(s, a)$) is incremented for that action. The threshold τ_{best} is used to prevent incrementing $C_{\text{streak}}(s, a)$ when an action only has the best Q-value for a very short time. There is also a condition to make sure the same action does not trigger this consecutively.

Once $\max_a C_{\text{streak}}(s, a)$ reaches τ_{streak}, that means that the policies have cycled enough since $\Delta_{\text{restricted}}$ last moved and it is time to move it once again. $InCycle(s, a)$ is a set containing all the actions that are assumed to be in the cycle that the policy is in. Requiring $C_{\text{streak}}(s, a)$ to be positive for all actions in $InCycle(s, a)$ is an attempt to prevent $\Delta_{\text{restricted}}$ from moving if it does not contain the Nash equilibrium.

4.2.6 "Shrink and Move Simplex"

$MoveSimplex(s)$ shown in Figure 4.4 mostly just resets variables, but there are two steps that are rather interesting. Step 2 allows $\Delta_{\text{restricted}}$ to be shrunk quickly if $\Delta_{\text{restricted}}$ has not been moved very far after a number of moves. t_{shrink} records the number of consecutive times that the Q-value interpolated for the center of $\Delta_{\text{restricted}}$ was larger than the other interpolated Q-values. If $t_{\text{shrink}} \geq \tau_{\text{shrink}}$, then $\Delta_{\text{restricted}}$ has seemed centered on the Nash equilibrium for a while, so PCWoLF-PHC could be shrunk quickly. f_{shrink} is the factor to multiply $C_{\epsilon}(s)$ by if $\Delta_{\text{restricted}}$ is to be shrunk quickly. Because $Center(s, a)$ is not the actual center of $\Delta_{\text{restricted}}$, changing ϵ would move the center of $\Delta_{\text{restricted}}$. Step 2a,iv handles making sure the center of $\Delta_{\text{restricted}}$ will not be changed when it is shrunk quickly.

Step 6 is where the center of $\Delta_{\text{restricted}}$ is actually moved. Step 6a checks if the center of $\Delta_{\text{restricted}}$ has the highest interpolated Q-value compared to the other ones recorded. If that is the case, then the amount $\Delta_{\text{restricted}}$ will be moved is decreased. f_{move} is the amount $\Delta_{\text{restricted}}$ is to move relative to the size of Δ_{policy}. f_{middle} is the fraction of f_{move} to move $\Delta_{\text{restricted}}$ if the center of $\Delta_{\text{restricted}}$ had the highest interpolated Q-value.

The mechanism used to move $Center(s, a)$ in Step 6b allows it to be moved in an infinite number of directions. The probability associated with any one action is increased by a value proportional to the difference of the minimum interpolated Q-value for the vertex of $\Delta_{\text{restricted}}$ associated with this action and the average over all vertices of $\Delta_{\text{restricted}}$. Step 6c makes sure $Center(s)$ plays each action at least probability η, so there is adequate exploration. η is only used to make sure that every action has at least some small probability of being played. If η is set to zero, the lack of exploration may cause PCWoLF-PHC to stop playing a certain action.

4.3 PCWoLF-PHC Simulations

Figure 4.6 shows two simulations of PCWoLF-PHC in self-play in Shapley's game. As can be seen by the center of the shrinking envelope, $\Delta_{\text{restricted}}$ centers quickly on the Nash equilibrium. $\Delta_{\text{restricted}}$ gets smaller rather slowly though. Different parameter values may cause the algorithm to use f_{shrink} to decrease the size of $\Delta_{\text{restricted}}$ more quickly. It is interesting to notice that in the simulation represented by Figures 4.6 b & c, one agent is able to shrink their simplex more quickly than the other. This shows that as one agent plays policies close to a mixed strategy Nash equilibrium, it makes it more difficult for the other to do the same.

If $\Delta_{\text{restricted}}$ is decreased too quickly, it can end up not containing the Nash equilibrium. Such a simulation would quite possibly not converge on the Nash equilibrium, even though steps in the algorithm were added to try and recover from such a state.

Another problem encountered is that the minimum values for the interpolated Q-values are noisy due to the α value being too large. Using a value for α that is too small would result in less noise, but the time required would be greatly increased.

PCWoLF-PHC seems to perform much better than the rest of the algorithms. Since PCWoLF-PHC requires the PHC layer to cycle a lot, values for α and δ that do not cycle quickly will cause PCWoLF-PHC to take a very long time. PCWoLF-PHC is fairly sensitive to the other parameters as well, but the algorithm could be changed to learn good values over time for most of them. This will be left to future work.

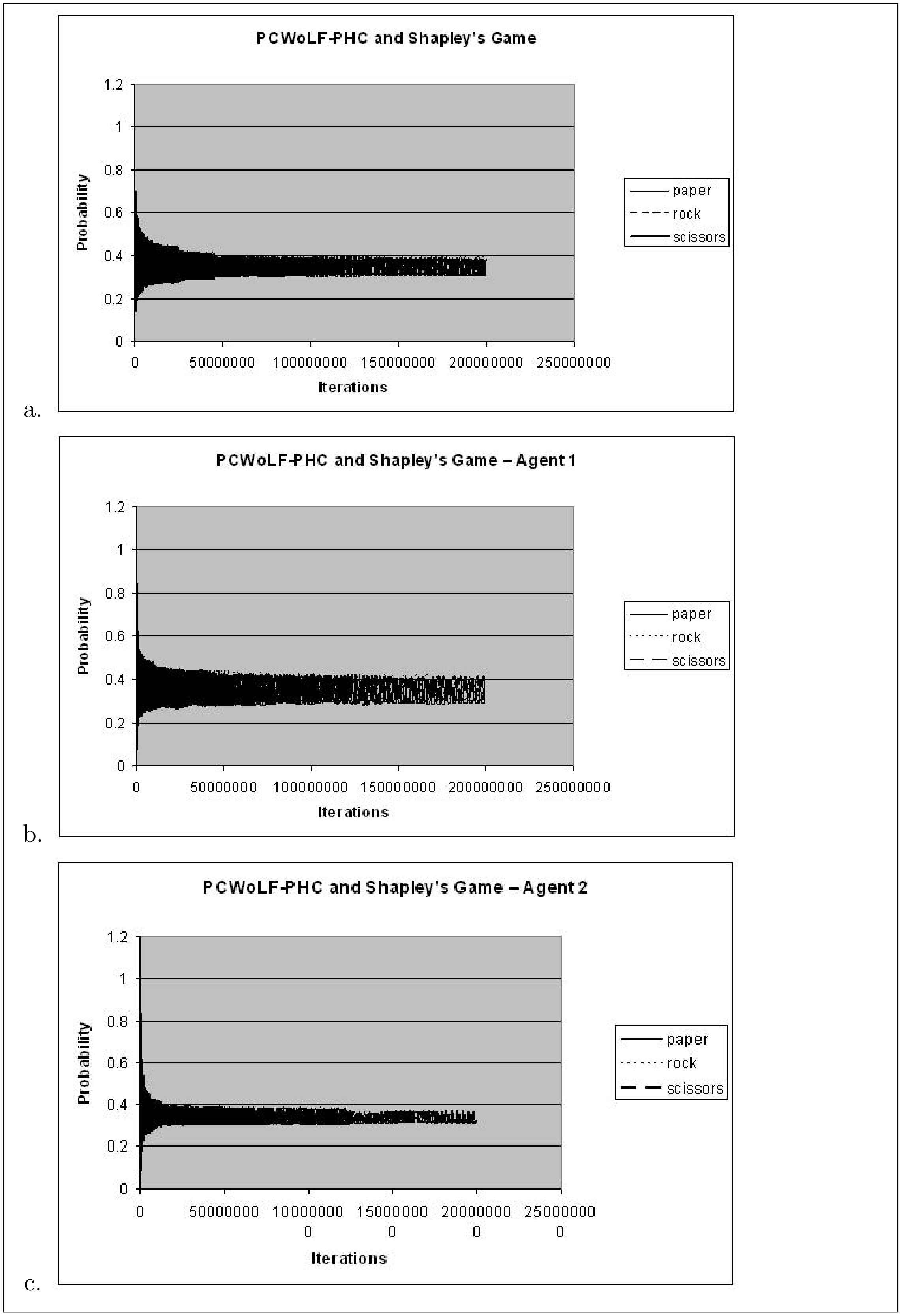

Figure 4.6: Two simulations of PCWoLF-PHC in self-play in Shapley's game. (b) and (c) show the policies of both agents in a simulation.

It can be argued that there exist parameters that ensure PCWoLF-PHC can converge in self-play on all games that WoLF-PHC will. If ϵ is set to 1 and ϵ_{offset} is set to ∞, then $Center(s, a)$ will never influence the policy that will actually be played. Another parameter setting that could be used to make PCWoLF-PHC act like WoLF-PHC is to set τ_{streak} to ∞. This essentially indicates that the PHC layer must cycle an infinite number of times before $\Delta_{\text{restricted}}$ will be moved.

4.4 Chapter Summary

Of the WoLF-PHC modifications tested, the only one that seems to work well is PCWoLF-PHC. PCWoLF-PHC tries to fix the cycling once it has happened, unlike the other modifications that try to avoid cycling. Simulations of PCWoLF-PHC in self play in Shapley's game look good, but convergence is slow. PCWoLF-PHC is fairly sensitive to its parameters, but we leave it to future work to simplify and improve the algorithm. Since PCWoLF-PHC was designed to fix WoLF-PHC after it diverges, it can be argued that PCWoLF-PHC will converge in self-play to the Nash equilibrium on any game that WoLF-PHC will.

In the next chapter, PCWoLF-PHC will be tested in self-play in various games. These tests are presented instead of an analysis of parameter sensitivity; if the algorithm works on a wide variety of games, we will consider it robust. These games include some in which WoLF-PHC converges to the Nash equilibrium and some for which it does not.

Chapter 5

Other Games

Chapter 4 evaluated PCWoLF-PHC in self-play in Shapley's game. This chapter evaluates PCWoLF-PHC in self-play in various other matrix games.

5.1 Matrix Games

Simulations of PCWoLF-PHC in self-play in seven matrix games will be examined. The seven matrix games that will be used in this chapter are listed in Figure 5.1.

Figure 5.1a shows the payoffs for Jordan's three-person matching pennies. One way to see how the extension from two-person to three-person was made is to think of the players standing in a circle and all facing clockwise. Each player compares their chosen action with the action chosen by the player they are facing. For all but the last player, if their actions are the same then they get a reward of 1 and otherwise a reward of -1 is received. For the last player, the payoff is 1 if the actions are different and -1 if they are the same. Using this framework, matching pennies can be extended to any number of players.

The Nash equilibrium for this game is to play each action half the time. As has been shown in [9] and Section 3.4.2, WoLF-PHC can converge to the Nash equilibrium when using the right parameters. Jordan's three-person matching pennies was chosen to show whether PCWoLF-PHC would help promote convergence using variable settings other than those that would cause WoLF-PHC to converge. Many simulations of WoLF-PHC

$$\left[\begin{array}{ccc} & H & T \\ H & (1,1,-1) & (-1,-1,-1) \\ T & (-1,1,1) & (1,-1,1) \end{array}\right] \qquad \left[\begin{array}{ccc} & H & T \\ H & (1,-1,1) & (-1,1,1) \\ T & (-1,-1,-1) & (1,1,-1) \end{array}\right]$$

(When player 3 chooses H) (When player 3 chooses T)

a. Jordan's Three-Person Matching Pennies

$$\left[\begin{array}{ccc} & Up & Down \\ Sun & (.95,-.95) & (1,-1) \\ Bottom & (1,-1) & (0,0) \end{array}\right] \qquad \left[\begin{array}{ccc} & C & D \\ C & (2,2) & (4,3) \\ D & (3,4) & (1,1) \end{array}\right] \qquad \left[\begin{array}{ccc} & H & T \\ H & (0,3) & (3,2) \\ T & (1,0) & (2,1) \end{array}\right]$$

b. Fighters & Bombers [15] c. Battle of the Sexes d. Tricky Game [9]

$$\left[\begin{array}{cccc} & P & R & S \\ P & (4,0) & (1,0) & (2,1) \\ R & (4,3) & (0,2) & (3,2) \\ S & (5,4) & (0,5) & (2,4) \end{array}\right] \qquad \left[\begin{array}{cccc} & P & R & S \\ P & (0,0) & (2,0) & (0,1) \\ R & (0,1) & (0,0) & (1,0) \\ S & (1,0) & (0,1) & (0,0) \end{array}\right]$$

e. Modified Shapley's Game f. Off-center Shapley's Game

$$\left[\begin{array}{cccc} & P & R & S \\ P & (0,0,1) & (1,0,1) & (0,1,1) \\ R & (0,0,0) & (0,0,0) & (1,1,0) \\ S & (1,0,0) & (0,0,0) & (0,1,0) \end{array}\right] \left[\begin{array}{cccc} & P & R & S \\ P & (0,1,0) & (1,0,0) & (0,0,0) \\ R & (0,1,1) & (0,0,1) & (1,0,1) \\ S & (1,1,0) & (0,0,0) & (0,0,0) \end{array}\right] \left[\begin{array}{cccc} & P & R & S \\ P & (0,0,0) & (1,1,0) & (0,0,0) \\ R & (0,0,0) & (0,1,0) & (1,0,0) \\ S & (1,0,1) & (0,1,1) & (0,0,1) \end{array}\right]$$

(When player 3 plays P) (When player 3 plays R) (When player 3 plays S)

g. Three-Person Shapley's Game

Figure 5.1: Matrix games for Chapter 5 simulations of PCWoLF-PHC.

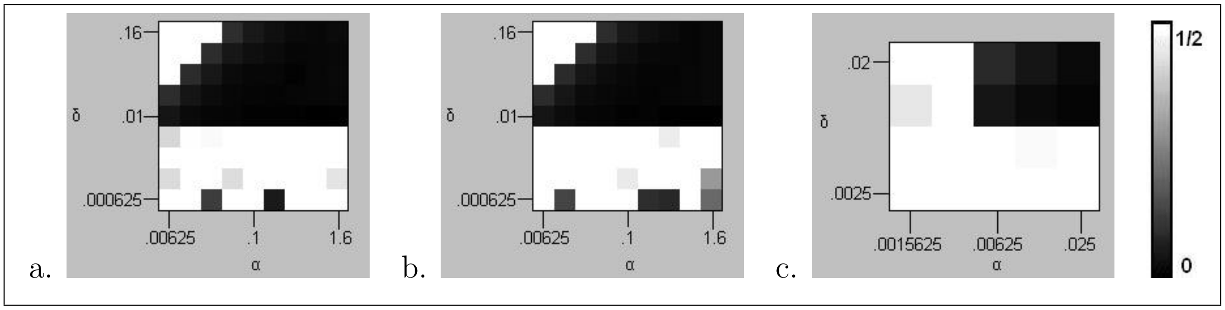

Figure 5.2: Simulations of WoLF-PHC in self-play playing Jordan's three-person matching pennies. The x-axis corresponds to the $\alpha(0)$ value, and the y-axis is the $\delta(0)$ value. The other parameters are $\alpha_{\text{offset}} = 100000$, $\delta_{\text{offset}} = 100$, $\frac{\delta_l}{\delta_w} = 3$, and $\epsilon = .1$. The simulations for (a) and (b) were run for 100 million iterations, and the simulations in (c) were run for a billion iterations. The values plotted are the largest difference from $\frac{1}{2}$ for any action probability for either agent during the last one percent of iterations. Figures 5.2a and b are obtained using different random seeds.

in self-play are summarized in Figure 5.2, which is repeated from Figure 3.12.

Figure 5.1b shows the payoffs for Fighters & Bombers [15]. This game was designed to model combat between a fighter and a bomber in World War II. The actions available to the fighter are sun attack (attack from above using the sun to make it hard to spot), and bottom attack (attack from below). The bomber can either look up or look down.

This game has a Nash equilibrium when the fighter performs a sun attack 20 out of 21 encounters and the bomber looks up 20 out of 21 encounters. This game was chosen because it has a mixed strategy Nash equilibrium very close to both agents playing a pure strategy. WoLF-PHC fails to converge in this game. Figure 5.3a shows how WoLF-PHC cycles in self-play without converging.

Figure 5.1c shows the payoffs for Battle of the Sexes. This game is modeling the choice of an activity for a couple. The action labeled "C" means they do their favorite activity, and "D" refers to their partner's favorite activity. They both would rather be together than not, but would also rather do their favorite activity than their partner's.

There are two pure strategy Nash equilibria corresponding to when they are doing the same activity ((C,D) and (D,C)), and a mixed strategy Nash equilibrium that occurs

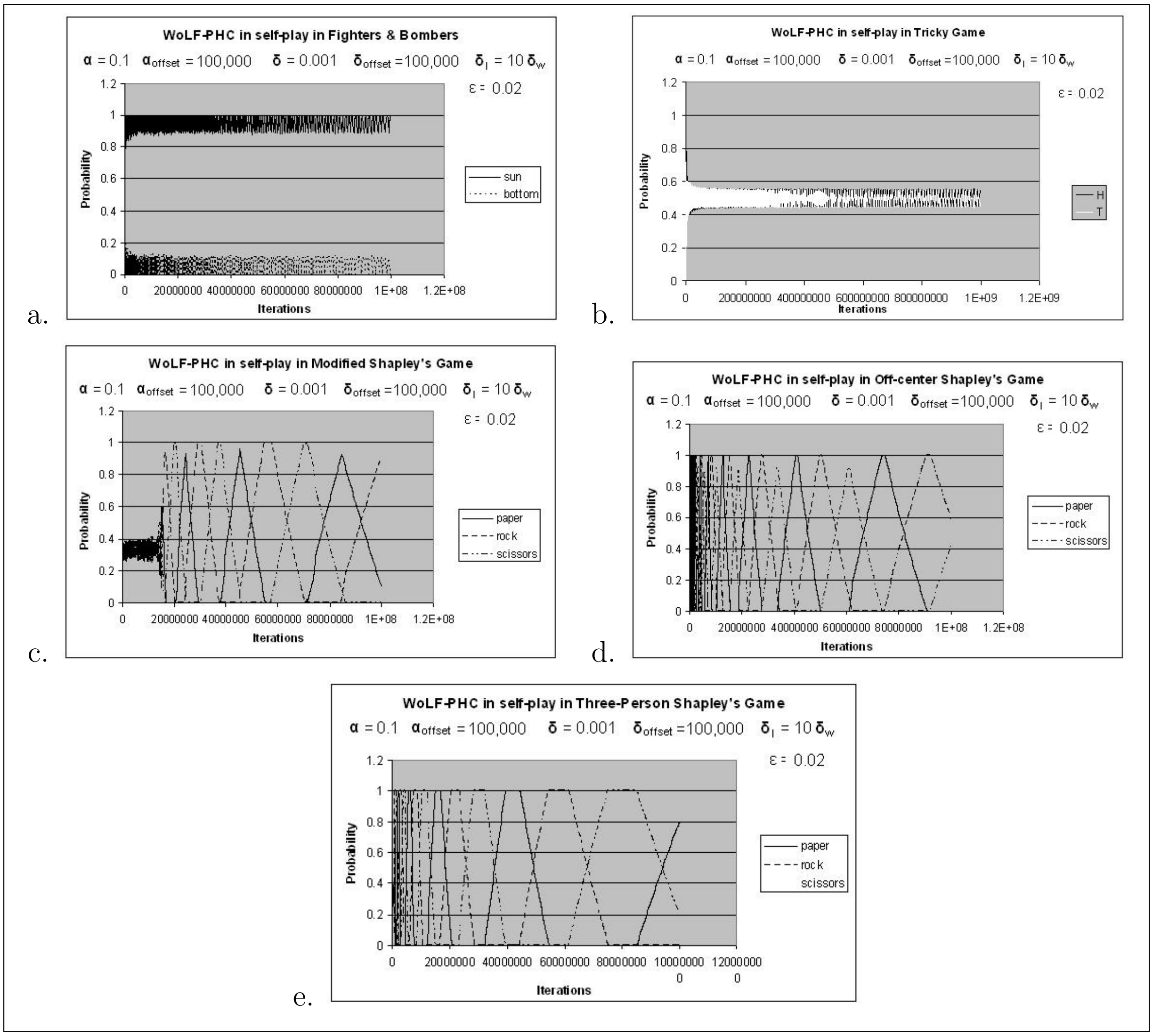

Figure 5.3: Simulation results for WoLF-PHC in self-play on various matrix games.

when both agents choose “C” 3 out of 4 times. This is the only game used in this Chapter that has pure strategy Nash equilibria. WoLF-PHC finds the pure strategy Nash equilibria with ease. Both agents start by playing “C” increasingly often, but randomness eventually drives one agent to play “D.”

Figure 5.1d shows the payoffs for Tricky game [9]. This game is similar to matching pennies or Fighters & Bombers in that one agent wants to select the same action of the other agent while the other agent tries to choose a different action than the first. The Nash equilibrium of this game is to play both actions half of the time. This game was chosen because it is a two-player two-action matrix game with the only equilibrium being to play uniformly random. Like Fighters & Bombers, WoLF-PHC fails to converge to the Nash equilibrium by maintaining a constant cycle. Figure 5.3b shows a simulation run for a billion iterations to help show that the cycle is not changing.

One thing that should be noted about Fighters & Bombers and Tricky game is that although there is a point that WoLF-PHC seems to stop converging, the parameters may be chosen such that the deviation from the Nash equilibrium is almost arbitrarily small. In other words, WoLF-PHC can get arbitrarily close to playing the Nash equilibrium but will not converge to it in any reasonable amount of time.

Figure 5.1e shows the payoffs for Modified Shapley’s game. The difference between this game and Shapley’s game was made to mirror the difference between Tricky game and a non-zero sum version of matching pennies. Notice that as the agents go around the best response cycle each agent’s reward increases by one and drops back to zero once along the cycle. The outcome that drops a given agent’s payoff to zero is one outcome away in the best response loop from where the other agent receives a payoff of zero.

There is one Nash equilibrium, and it occurs when both agents choose their actions uniformly random. This game was chosen because it is the combination of two games that WoLF-PHC has difficulties with. WoLF-PHC does not converge to the Nash equilibrium in self-play in this game. Figure 5.3c shows a simulation of WoLF-PHC in self-play in Modified Shapley’s game. It looks very similar to results from Shapley’s game, but the agent tends to not play paper as much. This can be seen by having its peak below the peaks for the other actions in any given cycle.

Figure 5.1f shows the payoffs for off-center Shapley's game. The only difference between this game and Shapley's game is that this game changes one payoff to a two. The Nash equilibrium for this game is to have the row agent play uniformly random, and for the column agent to play "P" and "S" with probability 2/5 and "R" with probability 1/5. This game was chosen for its asymmetric Nash equilibrium. WoLF-PHC fails to converge in self-play to the Nash equilibrium in this game. Figure 5.3d shows a simulation of WoLF-PHC in self-play in off-center Shapley's game. Like Modified Shapley's game, one action's peak probability does not reach as high as the other two.

Figure 5.1g shows the payoffs for three-person Shapley's game. This game uses a similar idea to Jordan's three-person matching pennies to extend Shapley's game to three players. If you imagine the agents standing in a circle, each player except the last will take the payoff they would receive if playing Shapley's game with the agent in front of them. Since the number of actions and the number of agents are not relatively prime, the last agent simply gets a payoff of 1 if it plays the same action that the first agent played. This is to prevent a situation where all agents can receive a payoff of 1. The Nash equilibrium for this game is to play uniformly random. This game was chosen because it is a game that will be harder for WoLF-PHC to converge on than Shapley's game, and because it is a three-player game. WoLF-PHC fails to converge in self-play to the Nash equilibrium in this game. Figure 5.3e shows the results of a simulation of WoLF-PHC in self-play in three-person Shapley's game. The biggest difference from other simulations shown is that the peaks take longer to change since information needs to propagate through another agent.

5.2 Simulations

Many simulations were run with PCWoLF-PHC in self-play in the matrix games shown. The following sections will describe how well PCWoLF-PHC worked on them.

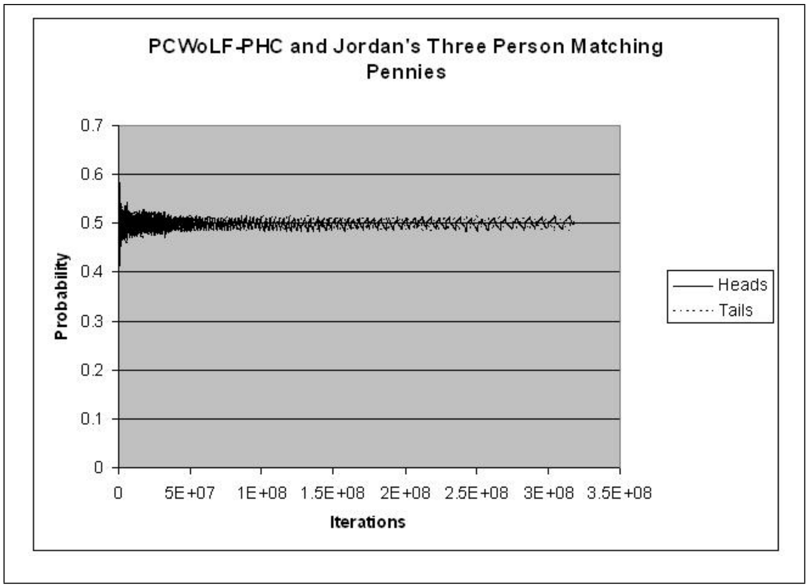

Figure 5.4: Simulation results for PCWoLF-PHC in self-play in Jordan's Three-person matching pennies.

5.2.1 Three-Person Matching Pennies

Since WoLF-PHC can converge to the Nash equilibrium in self-play with certain parameters, it is more interesting to see how PCWoLF-PHC works when using parameter values that cause WoLF-PHC to diverge. Figure 5.4 shows a simulation for a set of parameters that would cause WoLF-PHC to diverge. Like simulations with Shapley's game, the simplex stays centered but decreases in size slowly.

5.2.2 Fighters and Bombers [15]

As implemented, PCWoLF-PHC did not seem to have any advantage over WoLF-PHC when playing Fighters & Bombers. In other words, PCWoLF-PHC performed poorly unless it was using parameter values for which WoLF-PHC works fairly well. WoLF-PHC does not converge to the Nash equilibrium in this game, and neither does PCWoLF-PHC. The algorithm has trouble moving the simplex a large distance without shrinking the simplex too much. Further simulations with different parameter values may help, but the algorithm was not designed to easily center the simplex over Nash equilibria close to pure strategies.

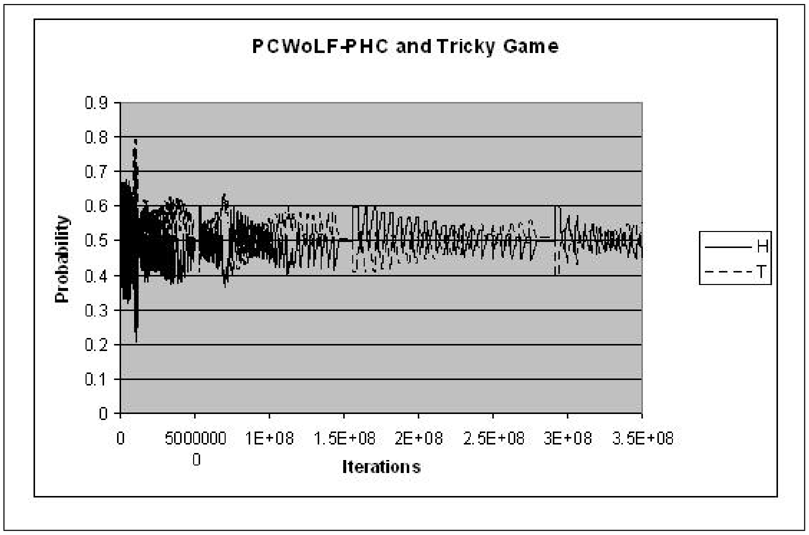

Figure 5.5: Simulation results for PCWoLF-PHC in self-play in Tricky game.

5.2.3 Battle of the Sexes

There was no difficulty in converging on this game. The policy would get as close to playing the pure strategy as the algorithm would let it. But the algorithm as currently implemented will not allow the policy to converge to a pure strategy completely. Some steps may be added to allow complete convergence, but this is left to future work.

5.2.4 Tricky Game [9]

PCWoLF-PHC did not perform as well as hoped on Tricky game. Figure 5.5 shows a simulation of PCWoLF-PHC in self-play in Tricky game. The way the simplex is sampled for interpolated Q-values is not sufficient for PCWoLF-PHC to see how the game dynamics are working. The agents will begin to look like they will converge, but then the simplex ends up too far away from the Nash equilibrium. The agents then both increase the size of their simplices, only to have the process repeat itself.

5.2.5 Modified Shapley's Game

Since this game was modeled after Tricky game, it should be no surprise that it is more difficult than Shapley's game for the PCWoLF-PHC algorithm. Figure 5.6 shows a simu-

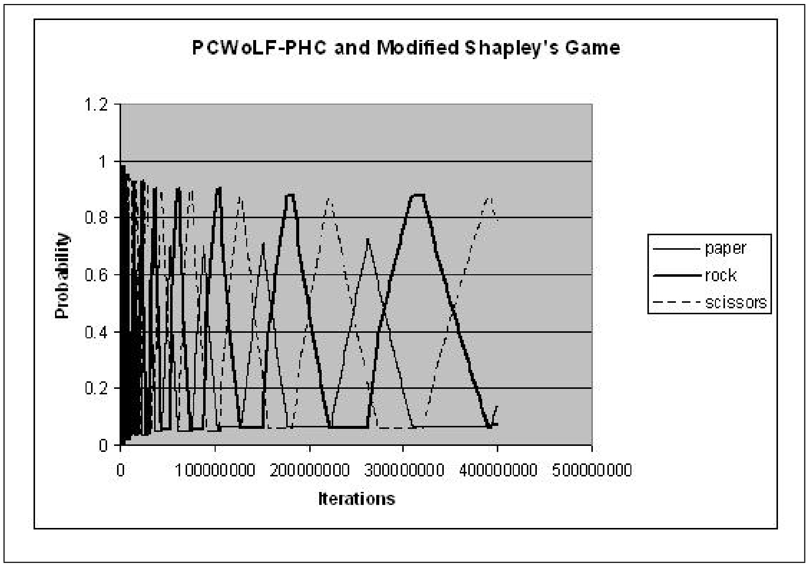

Figure 5.6: Simulation results for PCWoLF-PHC in self-play in Modified Shapley's game.

lation using PCWoLF-PHC in self-play in Modified Shapley's game. The parameters may be set such that it looks like it is working, but the time required to run the simulation long enough to make sure would be infeasible. Most simulations that were run had similar problems as simulations with Tricky game.

5.2.6 Off-center Shapley's Game

Figure 5.7 shows a simulation of PCWoLF-PHC in self-play in off-center Shapley's game. A similar problem to the one encountered in Modified Shapley's Game is also found with simulations using this game. There may be parameter settings that make PCWoLF-PHC converge more quickly, but the algorithm has a tendency to lose sight of the equilibrium.

5.2.7 Three-Person Shapley's Game

Simulations with this game act like simulations with Shapley's game, but they take much longer to converge due to another agent.Figure 5.8 shows a simulation of PCWoLF-PHC in self-play in three-person Shapley's game. It seems to be converging, but it is extremely slow.

Due to the extra player, the Q-values have more time to update before the PHC layer

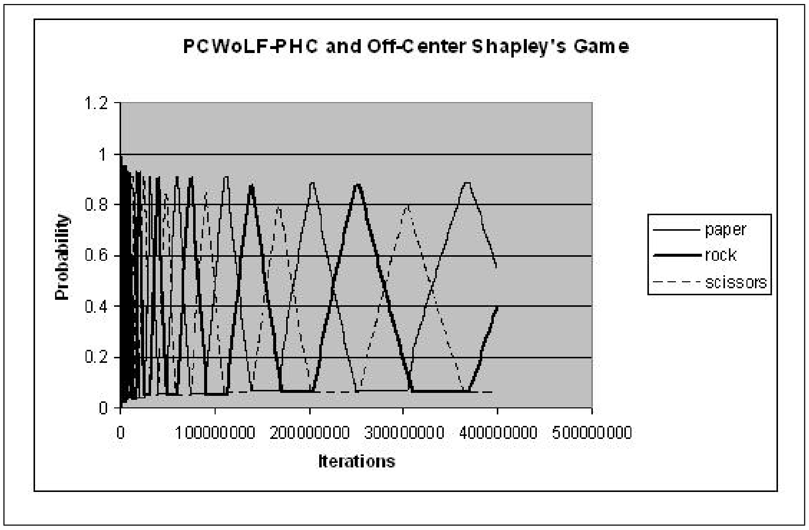

Figure 5.7: Simulation results for PCWoLF-PHC in self-play in off-center Shapley's game.

moves the policy. Therefore, the minimum interpolated Q-values may be more accurate estimates than when compared with simulations with Shapley's game.

5.3 Chapter Summary

PCWoLF-PHC does help performance in some games, but still does not converge to the Nash equilibium in all the games tested. When it seems PCWoLF-PHC may be converging, convergence happens very slowly. PCWoLF-PHC does not help with the problems WoLF-PHC has with games like Tricky game. Sampling the interpolated Q-values for more policies within the simplex should help PCWoLF-PHC center the simplex more easily, but other mechanisms may be required to converge faster.

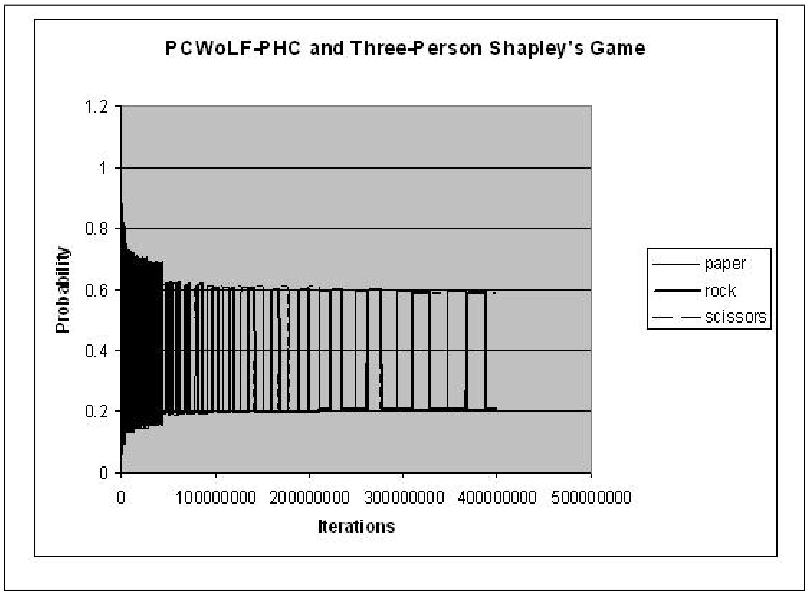

Figure 5.8: Simulation results for PCWoLF-PHC in self-play in Three-person Shapley's game.

Chapter 6

Conclusions

WoLF-PHC works quite well in self-play in many matrix games. PHC and WoLF-PHC have difficulties with Shapley's game and tend to diverge from the Nash equilibrium. This trend was seen over every set of parameters tried.

6.1 Modifications

Simple variations of WoLF-PHC have failed, seeming to show that this tendency to cycle is not a simple one to remove. The variations that were tested include modifying perceived rewards, modifying how the policy is updated, modifying the average policy calculation, and splitting up policy space and using different sets of variables for each section.

By contrast, PCWoLF-PHC works fairly well on Shapley's game. It also improves on WoLF, albeit slightly, in other games as well. However, although PCWoLF-PHC prevents divergence fairly well, it converges rather slowly.

When PCWoLF-PHC works, it does so because of a number of factors. One important factor is to use learning rates such that the Q-values reflect average reward enough that the minimum interpolated Q-values approximate minimum payoff against policies the opponent tends to play. The PHC policy update mechanism was designed for gradient ascent and not to get good estimates of average reward, so using learning rates that cause the PHC layer to get good estimates of Q-values is essential for PCWoLF-PHC. Nash equilibria that are symmetric with respect to the actions being played (i.e., playing random) and symmetric rewards help the relative accuracy of minimum interpolated Q-values. Asymmetric Nash equilibria and asymmetric rewards cause problems by adding asymmetric noise to the Q-values.

6.2 Future Work

Extensive form games and POMDPs can be used to model various important problems that cannot be accurately represented in a matrix game. Finding variants of existing algorithms that work well in these domains may prove to be a valuable contribution.

PCWoLF-PHC uses the insight that an approximation for the Q-value of a mixed policy can be calculated by a simple dot product of the vector of Q-values of pure strategies and the vector representing the mixed policy. Building on this idea, it can be seen that it is just a simple linear interpolation and the Q-values actually define an $N-1$ degree vector space, where N is the number of actions. The vector space changes as the Q-values change. An algorithm could use these spaces by intersecting them and storing a faceted representation of the minimum interpolated Q-values for every probabilistic policy. Of course, a certain amount of relaxation over time for the minimum Q-values values may be necessary.

A potentially useful change to PCWoLF-PHC is to choose a random set of policies within the simplex and keep track of the minimum interpolated Q-values for those. The randomly chosen policy with largest minimum interpolated Q-value will be an estimate for the agent's part of the Nash equilibrium. After repeating the process a number of times, the simplex can begin to shrink and will be centered on the average of the best random policies. As the simplex is shrunk, the random policies will be chosen with greater density around the Nash equilibrium and the estimate should get better. The standard deviation of the best random policies could also be used in determining the size of the simplex.

One problem with PCWoLF-PHC is its dependence on Q-values; a related problem is that the PHC layer moves its policy before Q-values give accurate measurements for average reward. It would help greatly if changes were made to the PHC layer such that its purpose would be getting good estimates for the average reward at the vertices of the simplex rather than simple gradient ascent. In other words, it would be good to keep more of a static policy until Q-values stop drifting. This would reduce the effect of noisy Q-values.

PCWoLF-PHC requires far too many parameters to be useful for practical purposes. Finding methods of learning good parameter values would help its usability and probably help its performance.

Bibliography

[1] J.-P. Allouche and T. Johnson. Narayana's cows and delayed morphisms. In *Cahiers du GREYC, Troisièmes Journées d'Informatique Musicale (JIM 96)*, volume 4, pages 2–7, 1996. Talks about Narayana's cows and the corresponding sequence problem.

[2] R. Axelrod. *The Evolution of Cooperation.* Basic Books, 1984. Describes many ideas relating to multiagent systems. In depth explanation of the prisoner's dilemma and established strategies for it.

[3] B. Bakker, F. Linaker, and J. Schmidhuber. Reinforcement learning in partially observable mobile robot domains using unsupervised event extraction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 938–943, 2002. ARAVQ is used to extract "events," then reinforcement learning is performed on the events.

[4] B. Banerjee and J. Peng. Adaptive policy gradient in multiagent learning. In *Second International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 686–692, 2003. This paper reviews the WoLF-iga algorithm, and presents the PDWoLF algorithm. This algorithm is guaranteed to converge to a nash equilibrium in 2 action bimatrix games.

[5] B. Banerjee and J. Peng. The role of reactivity in multiagent learning. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 538–545, 2004. The authors of this paper experiment with the "Exploiter" algorithm presented by Chang and Kaelbling that was developed to beat Q-learners

in repeated play games. WoLF-PHC is compared, among others. They also point out problems with the classification system Chang and Kaelbling presented.

[6] B. Banerjee, S. Sen, and J. Peng. Fast concurrent reinforcement learners. In *Seventeenth International Joint Conference on Artificial Intelligence*, pages 825–830, 2001. Discusses learning in multiagent situations, and the minimax-q algorithm is investigated. They propose a new algorithm that they call minimax-SARSA and show it to out perform minimax-q in certain general-sum domains.

[7] J. Baxter and P. L. Bartlett. Reinforcement learning in POMDP's via direct gradient ascent. In *17th International Conf. on Machine Learning*, pages 41–48. Morgan Kaufmann, San Francisco, CA, 2000. A reinforce-like algorithm named GPOMDP is presented and discussed.

[8] M. Bowling. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems 17*, 2004. Introduces the GIGA-WoLF algorithm and proves regret bounds and shows convergence in a limited class of games.

[9] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. In *Artificial Intelligence*, volume 136, 2002. This paper presents the Win or Learn Fast (WoLF) principle. A number of algorithms are discussed and WoLF-iga and WoLF-PHC are presented.

[10] M. Bowling and M. Veloso. Scalable learning in stochastic games. In *AAAI Workshop Proceedings on Game Theoretic and Decision Theoretic Agents*, 2002. This paper discusses results of a WoLF algorithm used on the game Goofspiel with self-play. The algorithm was made using the ideas of WoLF, tile coding, and policy gradient ascent.

[11] D. Braziunas and C. Boutilier. Stochastic local search for pomdp controllers. In *Nineteenth National Conference on Artificial Intelligence*, pages 690–696, 2004. Discusses problems with gradient based methods to solving POMDPs. A new stochastic local search method is proposed, and simulations show this method to be superior than gradient ascent.

[12] C. Camerer, T.-H. Ho, and J. Chong. A cognitive hierarchy theory of one-shot games. In *Quarterly Journal of Economics*, August, 2004. Mentions fictitious play and Shapley's class of games that cause cycles.

[13] C. F. Camerer and M. Knez. Increasing cooperation in prisoner's dilemmas by establishing a precedent of efficiency coordination in games. In *Social Science Working Paper 1080*. California Institute of Technology, January 2000. This paper discusses various topics relating to the Prisoner's Dilemma. It talks about how always defecting is the only stable strategy, but that variations can make cooperation fairly stable.

[14] A. R. Cassandra. Pomdps for dummies. Technical report, Deptartment of Computer Science, Brown University. http://www.cs.brown.edu/research/ai/pomdp/tutorial/pomdp-background.html, 1999. Gives and introduction to POMDPs and discusses their background. Discusses algorithms for POMDPs presented by various people.

[15] J. L. Casti. *Five Golden Rules: Great Theories of 20th-Century Mathematicians*. John Wiley & Sons Inc., New York, New York. 1997. This book is cited for the matrix game Fighters and Bombers.

[16] Y.-H. Chang and L. P. Kaelbling. Playing is believing: The role of beliefs in multi-agent learning. In *In Neural Information Processing Systems*, pages 1483–1490, Vancouver, Canada, 2001. A new classification is given for learning algorithms based on what the algorithm 'believes' about the opponent. They create an algorithm to try and trick the opponent by using 'decoy' policies.

[17] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *National Conference on Artificial Intelligence*, pages 746–752, 1998. This paper examines reinforcement learning and its application to multiagent systems. It looks at different types of Q-learning agents, and compares them.

[18] V. Conitzer and T. Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *20th International Conference on Machine Learning*, pages 83–90, 2003. This paper gives an algorithm for multiagent learning. This algorithm requires the ability to

precompute an equilibrium strategy and to be able to see the other agents actions. It also needs to be able to compute an equilibrium beforehand.

[19] J. W. Crandall and M. A. Goodrich. Establishing reputation using social committment in repeated games. In *AAMAS-04 Workshop on Learning and Evolution in Agent Based Systems.*, July 20, 2004, New York. Uses social committment and social utility functions to show how reputation can be used to improve performance in repeated social dilemmas.

[20] J. W. Crandall and M. A. Goodrich. Multiagent learning during on-going human-machine interactions: The role of reputation. In *In AAAI Spring Symposium: Interaction between Humans and Autonomous Systems over Extended Operation*, March 22-23, 2004, Standfor, California. This paper uses the idea of reputation, the expected pattern of behavior, to improve human-machine interactions. The idea of a reputational equilibrium is presented. A study was done involving people playing the prisoner's dilemma against fictitious play and again against a satisficing algorihtm.

[21] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. In *Games and Economic Behavior*, volume 29, pages 79–103, 1999. Presents an algorithm for repeated play games called "multiplicative weights" that is based on the "weighted majority algorithm" proposed by Littlestone and Warmuth.

[22] S. W. Hasinoff. Reinforcement learning for problems with hidden state. Technical report, Deptartment of Computer Science, University of Toronto. http://www.cs.utoronto.ca/∼hasinoff/pubs/hasinoff-rlhidden-2002.pdf, 2002. Discusses the theory behind POMDPs and gives a brief review of the algorithms presented for POMDPs.

[23] J. Hofbauer and E. Hopkins. Learning in perturbed asymmetric games. In *Journal of Economic Literature*, May, 2004. Discusses Nash equilibria and stability. Mentions fictitious play and Shapley's game.

[24] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Fifteenth International Conferece on Machine Learning*, pages

242–250, July 24-27, 1998. This paper presents an algorithm based on Littman's Q-learning algorithm, but extends it to allow for opponents that use mixed strategies. Properties of this algorithm are discussed.

[25] R. Karandikar, D. Mookherjee, D. Ray, and F. Vega-Redondo. Evolving aspirations and cooperation. In *Journal of Economic Theory*, volume 80, pages 292–331, 1998. Uses aspirations to help get "good enough" solutions and help foster cooperation.

[26] R. Khoussainov. Towards well-defined multi-agent reinforcement learning. In *Artificial Intelligence: Methodology, Systems, and Applications, 11th International Conference, Varna, Bulgaria*, pages 399–408, September 2-4, 2004. Khoussainov Demonstrates that a very broad class of Multiagent Reinforcement Learning problems are equivalent to POMDPs.

[27] J. W. Lawson and D. H. Wolpert. The design of collectives of agents to control non-markovian systems. In *Eighteenth national conference on Artificial intelligence*, pages 332–337, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. Groups of agents are used to learn POMDPs. They do this by extending the COllective INtelligence (COIN) to POMDPs.

[28] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. In *IEEE Symposium on Foundations of Computer Science*, pages 256–261, 1989. Discusses the weighted majority algorithm. It weights various predictors, and multiplies their weight by a constant if the prediction is incorrect.

[29] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 157–163, 1994. Descriptions of the markov framework are given in which opposing agents interact in an environment with probabilistic transition functions.

[30] M. L. Littman and P. Stone. A polynomial-time nash equilibrium algorithm for repeated games. In *Decision Support Systems*, volume 39, pages 55–66, 2005. This paper presents an algorithm for computing the Nash Equilibrium for an average-payoff

repeated bimatrix game. Finite state machine and a counting-node extension are the ways the represent strategies.

[31] J. F. Nash. Non-cooperative games. In *Annals of Mathematics*, volume 54, pages 286–295, 1951. This is probably Nash's most often cited work. It is the Seminal paper on Nash equilibria.

[32] J. J. O'Connor and E. F. Robertson. Leonhard euler. *http://www-gap.dcs.st-and.ac.uk/∼history/Biographies/Euler.html*, School of Mathematics and Statistics, University of St Andrews, Scotland. 1998. A biography for Leonhard Euler. Includes various achievements in mathematics.

[33] A. Odgaard and B. K. Nielsen. A visual implementation of fortune's voronoi algorithm. Technical report, Deptartment of Computer Science, University of Copenhagen. http://www.diku.dk/hjemmesider/studerende/duff/Fortune, 2001. Gives a definition of Voronoi polygons and a picture of them.

[34] T. J. Perkins. Reinforcement learning for pomdps based on action values and stochastic optimization. In *Eighteenth national conference on Artificial Intelligence*, pages 199–204, 2002. Reinforcement learning is applied to POMDPs. Hill climbing methods are examined as well as Monte Carlo and Sarsa methods.

[35] B. Price and C. Boutilier. Implicit imitation in multiagent reinforcement learning. In *The Sixteenth International Conference on Machine Learning*, pages 325–334, June 27-30, 1999. This paper deals with model extraction from observing other agents to help improve learning. This is a "less direct form of teaching" than traditional communication.

[36] S. Saha, S. Sen, and P. S. Dutta. Helping based on future expectations. In *Second International Joint Conference on Autonomous Agents and Multiagent Systems, Melbourne, Australia*, pages 289–296, July, 2003. This paper adds future expectations to the analysis of self-interested agents.

[37] T. Sandholm and R. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. In *Biosystems, Special Issue on the Prisoner's Dilemma*, volume 37,

pages 147–166, January, 1995. This paper dicusses the iterated prisoner's dilemma, and developed some q-learners to play against established strategies.

[38] H. Santana, V. Corruble, and B. Ratitch. Multi-agent patrolling with reinforcement learning. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1122–1129, 2004. They discuss the difficulties in turning patrolling into a reinforcement learning problem. A number of simulations are run and the results are analyzed.

[39] M. Sekaran and S. Sen. To help or not to help. In *Seventeenth Annual Conference of the Cognitive Science Society, Pittsburgh, Pennsylvania*, pages 736–741, July, 1995. This paper uses the idea of reciprocal behavior to try to help self-interested agents cooperate. They futher show in their simulations that agents who do not help other agents perferm worse overall than ones who reciprocate.

[40] S. Sen, S. Airiau, and R. Mukherjee. Towards a pareto-optimal solution in general-sum games. In *Second Intenational Joint Conference on Autonomous Agents and Multiagent Systems, Melbourne, Australia*, pages 153–160, July, 2003. Looking to future interaction, the agents in the simulations of this paper could increase payoffs from a Nash equilibrium to a Pareto-optimal solution.

[41] S. Singh, M. Kerns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 541–548, 2000. Gives the IGA algorithm for two-player, two-action games.

[42] S. S. Skiena. Voronoi diagrams. In *The Stony Brook Algorithm Repository.* Department of Computer Science, State University of New York, http://www.cs.sunysb.edu/∼algorith/files/voronoi-diagrams.shtml, 2001. Description of Voronoi polygons and algorithm to create them.

[43] J. L. Stimpson and M. A. Goodrich. Learning to cooperate in a social dilemma: A satisficing approach to bargaining. In *International Conference on Machine Learning*, pages 728–735, Washington D.C., USA, August 2003. This paper introduces the

multi-agent social dilemma (MASD) and uses the ideas of satisficing and the Nash bargaining solution to solve it.

[44] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. In *Autonomous Robotics*, volume 8, no. 3, pages 345–383, July, 2000. This survey describes how multiagent systems came to be, and some of the types of problems for which it is well suited. It discusses some reasons to use multiagent systems and gives some examples.

[45] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* Cambridge, Massachusetts: MIT Press/Bradford Books., 1998. Referenced by Thomas Perkins for the reinforcement learning algorithm they proposed for MDPs.

[46] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Tenth International Conference on Machine Learning*, pages 330–337, 1993. This paper examines the tradoffs in using communication to speed up learning rates among multiple agents. He gives three types of data that can be communicated between agents, and compares them using a case studies.

[47] J. Vidal. Learning in multiagent systems: An introduction from a game-theoretic perspective. In *In Adaptive Agents: LNAI 2636*, Springer Verlag, 2003. Discusses several game theory topics. Talks about CLRI theory and how it can be used to model systems.

[48] C. Watkins. *Learning from Delayed Rewards.* PhD thesis, Department of Psychology, King's College, Cambridge, UK, 1989. This is the first paper on Q-learning.

[49] C. Watkins and P. Dayan. Q-learning. In *Machine Learning*, volume 3, pages 279–292, 1992. This is the convergence proof for Q-learning.

[50] G. Weiss. *Learning in multiagent systems.* MIT Press, Cambridge, MA, 1999. This discusses multiagent learning, and how it relates to distributed AI and machine learning. It discusses different learning algorithms and how they work in a distributed environment.

[51] E. W. Weisstein. Voronoi diagram. *From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/VoronoiDiagram.html*, 1997. Gives a definition of Voronoi polygons and a picture of them.

[52] M. Wiering and J. Schmidhuber. Solving POMDPs with levin search and EIRA. In *Machine Learning: Procceedings of 13th International Conference*, Bari, Italy, 1996. Levin's universal search through program space is discussed as a way to solve POMDPs. Properties and optimality of the Levin search are discussed.

[53] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1997. HQ-learning is presented as an algoritm designed to learn POMDPs. HQ-learning is a hierarchical extension to Q(λ)-learning.

[54] M. Wooldridge. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* The MIT Press, Cambridge, Massachusetts, 1999. Gives an overview of DAI and MAS. Cited for some of its definitions.

[55] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 925–928, 2003. Introduced the Generalized Infinitesimal Gradient Ascent (GIGA) algorithm.

Appendix A

Validation of learning rate equations

C. Watkins proved that Q-learning would converge under certain conditions [49]. One of these conditions involves the speed at which the learning rates decay. To justify the decay equations used in this thesis, we will show that they follow the conditions specified by C. Watkins.

To show that these equations follow the criteria given by Watkins in the convergence proof for Q-learning [49], we must show that

$$\exists C \in \Re, \lim_{t \to \infty} \sum_{i=0}^{t} \alpha(i)^2 \leq C, \tag{A.1}$$

and

$$\forall C \in \Re, \lim_{t \to \infty} \sum_{i=0}^{t} \alpha(i) > C. \tag{A.2}$$

First, the variable $\alpha(0)$ can be shown to have no influence on convergence or divergence for the previous equations by

$$\begin{aligned} \lim_{t\to\infty} \textstyle\sum_{i=0}^{t} \left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+t}\right)^2 \leq C \quad &\Leftrightarrow \quad \lim_{t\to\infty} \textstyle\sum_{i=0}^{t} \alpha(0)^2 \left(\frac{\alpha_{\text{offset}}}{\alpha_{\text{offset}}+t}\right)^2 \leq C \\ &\Leftrightarrow \quad \alpha(0)^2 \lim_{t\to\infty} \textstyle\sum_{i=0}^{t} \left(\frac{\alpha_{\text{offset}}}{\alpha_{\text{offset}}+t}\right)^2 \leq C, \end{aligned}$$

$$\begin{aligned} \lim_{t\to\infty} \textstyle\sum_{i=0}^{t} \frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+t} > C \quad &\Leftrightarrow \quad \lim_{t\to\infty} \textstyle\sum_{i=0}^{t} \alpha(0) \frac{\alpha_{\text{offset}}}{\alpha_{\text{offset}}+t} > C \\ &\Leftrightarrow \quad \alpha(0) \lim_{t\to\infty} \textstyle\sum_{i=0}^{t} \frac{\alpha_{\text{offset}}}{\alpha_{\text{offset}}+t} > C. \end{aligned}$$

Since $\alpha(0)$ can be pulled out, we see that given any value for C that satisfies one of the equations with $\alpha(0) = 1$, we can find another value for C that will work for $0 \leq \alpha(0) \leq 1$.

Since setting the value for $\alpha(0)$ to 1 will not affect convergence, the second condition (A.2) can be seen easily by recalling that

$$\lim_{j\to\infty}\sum_{i=1}^{j}\frac{1}{i}=\lim_{j\to\infty}\sum_{i=0}^{j}\frac{1}{1+i}=\infty,$$

and

$$\forall\alpha_{\text{offset}}\geq 1,\forall i\geq 0,\alpha(i)=\frac{\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\geq\frac{1}{(1+i)}.$$

Since each term in the decay equation is greater than its counterpart in the other, we see that the decay equation must also diverge to ∞. For an arbitrary value of $\alpha(0)$ in the range $(0,1]$, we note that the sum will only be multiplied by $\alpha(0)$ and $\infty * \alpha(0)=\infty$.

To show that the first condition (A.1) holds, we first note that

$$\lim_{t\to\infty}\sum_{i=0}^{t}\frac{1}{(1+i)^2}=\lim_{t\to\infty}\sum_{i=1}^{t}\frac{1}{i^2}=\frac{\pi^2}{6}$$

as proven by Leonhard Euler [32].

Notice that for $i\geq 0$, $\alpha_{\text{offset}}\geq 1$, and $\alpha(0)>0$ the decay equation is monotonically decreasing as i increases. This means that the sum of the first n values will always be less than n times the first value. Notice that if i is 0, then

$$\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\right)^2=\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}}\right)^2=\alpha(0)^2\left(\frac{\alpha_{\text{offset}}}{\alpha_{\text{offset}}}\right)^2=\alpha(0)^2\left(\frac{1}{1}\right)^2=\alpha(0)^2\left(\frac{1}{1+i}\right)^2.$$

Because the decay equation is monotonically decreasing, we can then see that

$$\sum_{i=0}^{\alpha_{\text{offset}}-1}\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\right)^2\leq\alpha_{\text{offset}}\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+0}\right)^2=\alpha_{\text{offset}}\left(\frac{\alpha(0)}{1+0}\right)^2=\alpha(0)^2\alpha_{\text{offset}}\left(\frac{1}{1+0}\right)^2,$$

and similarly

$$\sum_{i=\alpha_{\text{offset}}}^{2\alpha_{\text{offset}}-1}\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\right)^2\leq\alpha(0)^2\alpha_{\text{offset}}\left(\frac{\alpha_{\text{offset}}}{\alpha_{\text{offset}}+\alpha_{\text{offset}}}\right)^2=\alpha(0)^2\alpha_{\text{offset}}\left(\frac{1}{1+1}\right)^2,$$

and in general

$$\sum_{i=n\alpha_{\text{offset}}}^{(n+1)\alpha_{\text{offset}}-1}\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\right)^2\leq\alpha(0)^2\alpha_{\text{offset}}\left(\frac{\alpha_{\text{offset}}}{\alpha_{\text{offset}}+n\alpha_{\text{offset}}}\right)^2=\alpha(0)^2\alpha_{\text{offset}}\left(\frac{1}{1+n}\right)^2.$$

Since this inequality is true for any value of n, we can sum over all values up to a certain value of n. Therefore, $\forall n\geq 0\in\mathbb{Z},\forall\alpha_{\text{offset}}\geq 1\in\mathbb{Z}$, and $\forall\alpha(0)>0\in\Re$,

$$\sum_{i=0}^{(n+1)\alpha_{\text{offset}}-1}\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\right)^2 \leq \alpha(0)^2\alpha_{\text{offset}} * \sum_{i=0}^{n}\left(\frac{1}{1+i}\right)^2.$$

It follows that

$$\lim_{n\to\infty}\sum_{i=0}^{(n+1)\alpha_{\text{offset}}-1}\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\right)^2 = \lim_{n\to\infty}\sum_{i=0}^{n}\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\right)^2,$$

and

$$\lim_{n\to\infty}\sum_{i=0}^{n}\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\right)^2 \leq \alpha(0)^2\alpha_{\text{offset}} * \lim_{n\to\infty}\sum_{i=0}^{n}\left(\frac{1}{1+i}\right)^2,$$

and finally

$$\alpha(0)^2\alpha_{\text{offset}} * \lim_{n\to\infty}\sum_{i=0}^{n}\left(\frac{1}{1+i}\right)^2 = \frac{\alpha(0)^2\alpha_{\text{offset}}\pi^2}{6}.$$

Therefore

$$\lim_{n\to\infty}\sum_{i=0}^{n}\left(\frac{\alpha(0)*\alpha_{\text{offset}}}{\alpha_{\text{offset}}+i}\right)^2 \leq \frac{\alpha(0)^2\alpha_{\text{offset}}\pi^2}{6}.$$

Therefore the first condition is satisfied as well, and the learning rate decay equations will allow Q-learning to converge according to the convergence proof given by C. Watkins. Since we know the equations will allow the Q-learning layer to converge, we can now decide on the parameter values to use in the decay equations.

Appendix B

WoLF-PHC Modifications, In Detail

Chapter 4 discussed PCWoLF-PHC in detail. This Chapter gives more detail on the modifications that did not perform well. Table B.1 gives a summary of a few of the modifications tested and their key properties.

In WoLF-PHC, the average policy is affected by the δ decay rate. WWoLF-PHC removes this dependence which will hopefully keep the average policy closer to playing random, and may aid in converging to playing random.

VWoLF-PHC uses a copy of the WoLF-PHC algorithm for each action. This can potentially help the agent remember the effects of past actions. The copy of the algorithm assigned to a specific action is only used if the current policy being used has that action played most often.

CWoLF-PHC and RPWoLF-PHC are similar to each other; both change how the reward they receive is used to update values. CWoLF-PHC uses the frequency of outcomes to determine when to alter reward values. RPWoLF-PHC modifies the reward value received when playing the action with decreasing probability (e.g., an action that does not have the highest Q-value) that is played most frequently, hereafter referred to as the *greatest falling action* (a_{gfa}).

PWoLF-PHC assumes that the policy had moved too far before the arg max shift, then tries to compensate for this by stepping the policy away from a_{gfa}.

The implementation and some results in self-play in Shapley's game will be shown for

Name	Key property
WWoLF-PHC	Weight the contribution of the current policy to the average policy by that of the delta value used in the current policy update.
VWoLF-PHC	Use a copy of the WoLF-PHC algorithm for each Voronoi cell in the probability hypersimplex created by using pure strategies as the lattice points. Each copy uses the same policy.
CWoLF-PHC	Decay the reward perceived by the algorithm by a function of the number of identical outcomes in a window of time.
RPWoLF-PHC	Decay the reward perceived when a_{gfa} is played.
PWoLF-PHC	Step the policy away from a_{gfa} after the policy update steps the policy toward the action with the highest Q-value.
PCWoLF-PHC	Move a simplex over policy space with interpolated Q-values sampled at the vertices to try and fix WoLF-PHC after it begins to cycle. Discussed in Chapter 4

Table B.1: WoLF-PHC modifications.

each modification.

B.1 Weighted WoLF-PHC (WWoLF-PHC)

After each iteration, WoLF-PHC updates the average policy, $\bar{\pi}$. This average policy is the average of every policy previously used. Since δ is decayed, later stages of learning require more iterations to be run to produce substantial changes in the policy. This biases the average policy towards the current policy for the agent.

WWoLF-PHC weighs the average policy in a way that removes its dependence on δ decay. When updating the average policy, the current policy is scaled by the current decayed δ as shown in step 2e in Table B.2.

When playing Shapley's game in self-play, WWoLF-PHC should keep the average policies closer to the Nash equilibrium. Notice that when cycling around a Nash Equilibrium the policies which were used while on the opposite side of the equilibrium will keep a larger influence on the average policy then when using WoLF-PHC.

This could help the algorithm to decide more accurately whether it is "winning" or not. The algorithm for this modification is given in Table B.2. Notice that the difference from WoLF-PHC occurs in step 2e. $C(s)$ now holds the sum of all the delta values used so far, and δ is used to scale the contribution of the current policy.

WWoLF-PHC does not perform well in Shapley's game. Figure B.1a shows the results of the sixteen simulations listed in Table 3.2. The shaded regions on the first two rows can be attributed to pseudoconvergence. Every other cell is white, indicating that the learned policy is maximally far from the Nash Equilibrium. Figure B.1b shows one of the simulations from Figure B.1a. Both agents' policies are shown, and the graph is typical of that of regular WoLF-PHC.

In summary, WWoLF-PHC does not converge in self-play in Shapley's game. It seems that removing the bias in the average policy when using WoLF-PHC did not have enough of an effect.

1. Let $\alpha, \delta_l > \delta_w$ be learning rates. Initialize,

$$Q(s,a) \leftarrow 0, \quad \pi(s,a) \leftarrow \frac{1}{|A_i|}, \quad \bar{\pi}(s,a) \leftarrow \frac{1}{|A_i|}, \quad C(s) \leftarrow 0.$$

2. Repeat,

   (a) From state s select action a with probability $\pi(s,a)$, with suitable exploration

   (b) Observing reward r and next state s'.

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

   (c) Set δ

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

   (d) Decay δ appropriately.

   (e) Update estimate of average policy, $\bar{\pi}$,

   $$\forall a' \in A_i \quad \bar{\pi}(s,a') \leftarrow \frac{C(s)}{C(s)+\delta}\bar{\pi}(s,a') + \frac{\delta}{C(s)+\delta}\pi(s,a'),$$
   $$C(s) \leftarrow C(s) + \delta.$$

   (f) Step π closer to the optimal policy w.r.t. Q. Same as PHC (2c).

Table B.2: Weighted WoLF-PHC algorithm for player i.

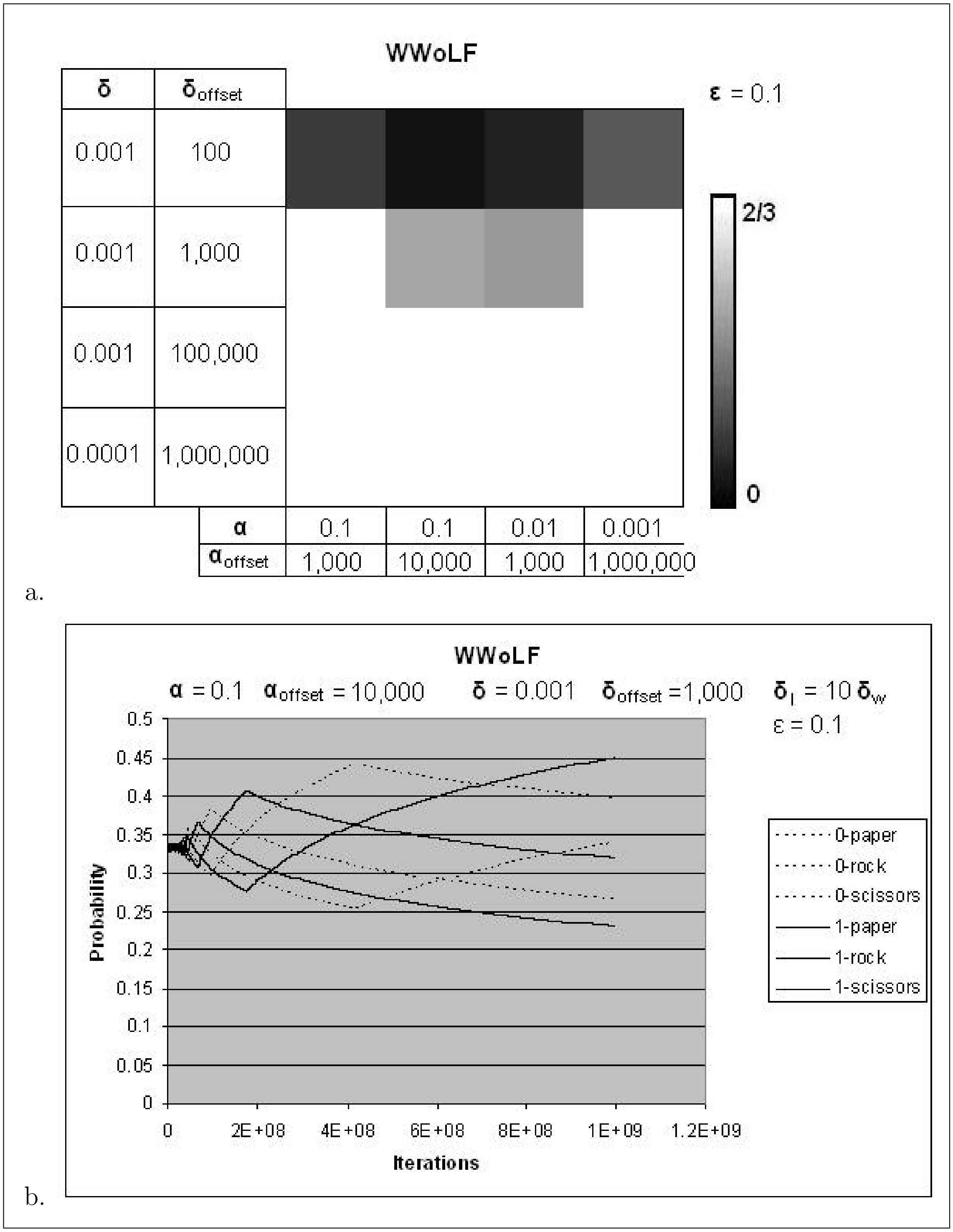

Figure B.1: (a) Results from 16 simulations using WWoLF-PHC. The value plotted is the largest distance from 1/3 for any action's probability in either agent's policy after 900 million iterations. (b) Graph of one of the simulations from (a) which shows divergence.

B.2 Voronoi WoLF-PHC (VWoLF-PHC)

When playing Shapley's game in self play, PHC and WoLF-PHC will change their behavior when their opponent changes which action is played most often. More precisely, policies start moving in a different direction (on average) when the opponent's most favored action changes. These changes correspond to when the opponent's policy moves from one Voronoi cell in probability space to another cell; in this description, the lattice points used to create the Voronoi decomposition correspond to pure strategies as shown in Figure 3.10b.

Remember from Section 3.3.6 that a Voronoi cell associated with point i is the set of all points closer to lattice point i than any other point in the lattice. In our case, each Voronoi cell contains all policies that correspond to a policy with a given action's probability greater than the others.

VWoLF-PHC uses a copy of the WoLF-PHC algorithm for each Voronoi cell in policy space, where the lattice points are the pure strategies. The agent will use the copy of the algorithm that is associated with the cell in which the agent's policy currently resides. This is done in an attempt to have the algorithm remember what happened the last time it played a similar policy. If this does improve the algorithm, then there is a tradeoff between memory usage and performance. The algorithm for VWoLF-PHC is given in Table B.3.

Like WWoLF, VWoLF does not perform well in Shapley's game. Figure B.2a shows the results from the 16 simulations listed in Table 3.2. Again, most regions in the figure are white meaning that policies were maximally far from the Nash equilibrium. Comparing Figure B.2a to Figure B.1a shows a little improvement, but still nothing that cannot be attributed to pseudoconvergence. Figure B.2b shows one of the simulations from Figure B.2a. This simulation shows pseudoconvergence ending around 800 million iterations.

The reason this algorithm does not work well is because it does not remember what happened for a long enough time. As soon as the agent's policy enters a new Voronoi cell, the Q-values quickly move to near the Q-values of the copy of the WoLF-PHC algorithm corresponding to the cell that the agent's policy just left. The algorithm essentially does nothing with the information stored in the multiple sets of Q-values.

1. Let $\alpha, \delta_l > \delta_w$ be learning rates. Initialize,

$$Q_{a'}(s,a) \leftarrow 0, \quad \pi(s,a) \leftarrow \frac{1}{|A_i|}, \quad C(s) \leftarrow 0.$$

2. Repeat,

   (a) From state s select action a with probability $\pi(s,a)$, with suitable exploration

   (b) Let $a_{\max} = \arg\max_{a'} \pi(s,a')$.

   (c) Observing reward r and next state s'.

$$Q_{a_{\max}}(s,a) \leftarrow (1-\alpha) Q_{a_{\max}}(s,a) + \alpha \left(r + \gamma \max_{a'} Q_{a_{\max}}(s',a') \right)$$

   (d) Update estimate of average policy, $\bar{\pi}$,

$$C(s) \leftarrow C(s) + 1$$

$$\forall a' \in A_i \quad \bar{\pi}(s,a') \leftarrow \bar{\pi}(s,a') + \frac{1}{C(s)}(\pi(s,a') - \bar{\pi}(s,a')).$$

   (e) Step π closer to the optimal policy w.r.t. Q.

$$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } \; a = \arg\max_{a'} Q_{a_{\max}}(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}$$

   Using a value for δ given by

$$\delta = \begin{cases} \delta_w & \text{if } \; \sum_{a'} \pi(s,a') Q_{a_{\max}}(s,a') > \sum_{a'} \bar{\pi}(s,a') Q_{a_{\max}}(s,a') \\ \delta_l & \text{otherwise} \end{cases}$$

Table B.3: Voronoi WoLF-PHC algorithm for player i.

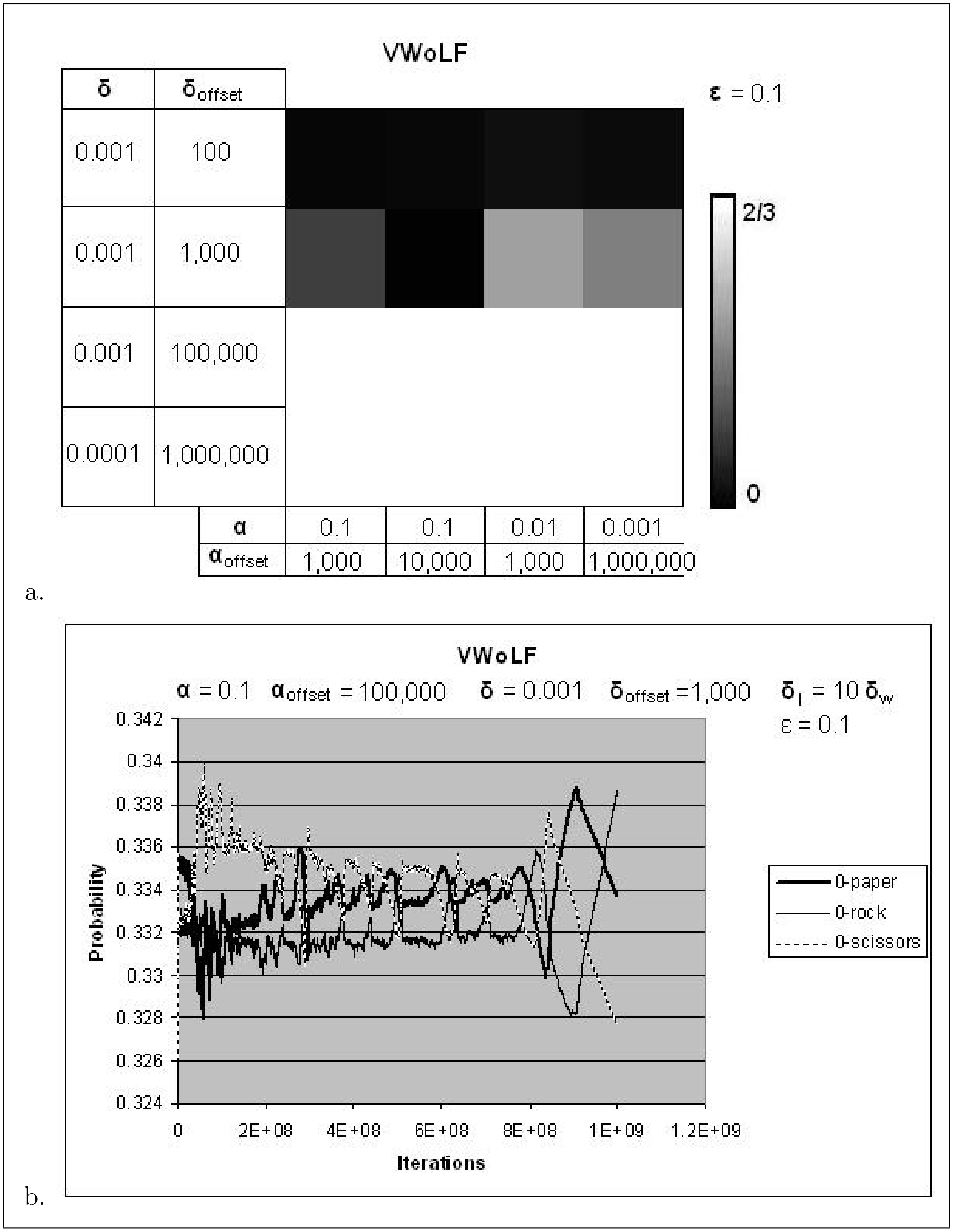

Figure B.2: (a) Results from 16 simulations using VWoLF-PHC. The value plotted is the largest distance from the Nash equilibrium for any action's probability in either agent's policy after 900 million iterations. (b) Graph of one of the simulations from (a) showing divergence.

B.3 Consecutive WoLF-PHC (CWoLF-PHC)

From the proof in Chapter 3, we see what occurs when an agent uses PHC in self-play in Shapley's game. Under these conditions, an agent moves too far toward a pure strategy while the other agent is adapting. The WoLF principle seems to be designed for this sort of problem, but does not completely solve the problem as implemented. However, WoLF-PHC does find the Nash equilibrium in the paper-rock-scissors game. The penalty of -1 in paper-rock-scissors aids in lowering Q-values once the opposing agent has changed its strategy. This allows the agent to conclude sooner that it is no longer winning and therefore shift to the larger value of δ.

CWoLF-PHC tries to mimic this behavior by decreasing the reward for consecutive identical outcomes. If the algorithm chooses the same action it chose on the previous iteration and the same reward was received, then that reward is treated as being less than what the reward actually was. This would be useful in helping the agent to be a little more cautious when things are going well. This means that Q-values do not shift as much which, in turn, makes it possible to decide sooner that it is losing.

To implement this modification, we need some mechanism that allows us to devalue consecutive rewards. One such mechanism is to devalue the reward based on the number of iterations since an arg max shift has occured. This mechanism alone is not sufficient to help WoLF-PHC to converge on Shapley's game without further modification. Consecutive outcomes are much less frequent when policies are near the Nash equilibrium; such outcomes increase in frequency as policies shift toward pure strategies. Thus, this mechanism would have minimal effect unless more information is used. Such information should include the number of iterations since a given outcome has happened, or the number of identical outcomes in a certain number of trials. Such information would allow this modification to have more effect when policies are closer to the Nash equilibrium.

To preserve the effect the original payoffs have on the Nash equilibrium, we can use two sets of Q-values: one set for *actual* Q-values and the other set for *decreased* Q-values. The policy change in PHC is based on the set of Q-values determined from the actual rewards.

Q-values based on the decreased rewards are used in WoLF's calculation to determine whether the agent is "winning." If this were not so, this algorithm would not converge in self-play to the Nash equilibrium on many matrix games. For some matrix games, a Nash equilibrium could be based on an indifference point (i.e., equality) of average rewards, which means that using the decreased Q-values would cause PHC to select an indifference point which is not a Nash equilibrium.

An algorithm that uses this mechanism is given in Table B.4. The information in $Q_{\mathrm{wl}}(s, a)$ are the modified Q-values that will be used in determining whether an agent is winning. The function $F(r_t, x)$ equals r_t if the action chosen is not the one currently favored by the Q-learning layer, and equals $r_t - p * (1 - \frac{po}{po+x})$ otherwise, where p is the maximum penalty parameter, x is the number of identical outcomes, and po determines how quickly the penalty increases as x increases. The penalty subtracted can never exceed p. The variable t is the number of trials run so far. The variable h in Table B.4 allows a specific window of the history to be examined for identical outcomes.

Figure B.3a and Figure B.3b show the results from the 16 simulations listed in Table 3.2 for two different values of p. Comparing Figure B.3a to Figure B.1a we see a slight decrease in performance. We should note that the simulations that differ were attributed to pseudoconvergence in Figure B.1a.

Because paper-rock-scissors has a penalty of 1, the parameter p in CWoLF was guessed to work well with a value of 1. Unfortunately, poor performance persists over all values of po.

Figure B.4 shows a simulation using CWoLF-PHC. The policies for both players are plotted, and the most notable difference from this mechanism is that it tends to have one agent move faster from playing random.

B.4 Reward Penalizing WoLF-PHC (RPWoLF-PHC)

Like CWoLF-PHC, RPWoLF-PHC tries to emulate what WoLF-PHC does with paper-rock-scissors. Consider this example of WoLF-PHC in self-play with paper-rock-scissors.

1. Let $\alpha, \delta_l > \delta_w$ be learning rates, $h \in \Re$ be the length of the history window, and F(r,x) be a given function. Initialize,

$$Q(s, a) \leftarrow 0, \quad Q_{\text{wl}}(s, a) \leftarrow 0, \quad \pi(s, a) \leftarrow \frac{1}{|A_i|}, \quad C(s) \leftarrow 0, \quad t \leftarrow 0.$$

2. Repeat,

   (a) From state s select action a_t with probability $\pi(s, a_t)$, with suitable exploration

   (b) Observing reward r_t and next state s', update Q and Q_{wl} as

$$Q(s, a_t) \leftarrow (1 - \alpha)Q(s, a_t) + \alpha \left(r_t + \gamma \max_{a'} Q(s', a') \right)$$

$$Q_{\text{wl}}(s, a_t) \leftarrow (1 - \alpha)Q_{\text{wl}}(s, a_t) + \alpha \left(F\left(r_t, \sum_{i=0}^{h-1} g(t - i) \right) + \gamma \max_{a'} Q_{\text{wl}}(s', a') \right)$$

   where $g(i) = \begin{cases} 1 & \text{if } r_i = r_t \text{ and } a_i = a_t \\ 0 & \text{otherwise.} \end{cases}$

   (c) Update estimate of average policy, $\bar{\pi}$,

$$C(s) \leftarrow C(s) + 1$$

$$\forall a' \in A_i \quad \bar{\pi}(s, a') \leftarrow \bar{\pi}(s, a') + \frac{1}{C(s)}(\pi(s, a') - \bar{\pi}(s, a')).$$

   (d) Step π closer to the optimal policy w.r.t. Q. Use the same method as PHC (2c), but select δ_w or δ_l using Q_{wl} according to

$$\delta = \begin{cases} \delta_w & \text{if } \sum_{a'} \pi(s, a')Q_{\text{wl}}(s, a') > \sum_{a'} \bar{\pi}(s, a')Q_{\text{wl}}(s, a') \\ \delta_l & \text{otherwise.} \end{cases}$$

   (e) $t \leftarrow t + 1$

Table B.4: Consecutive WoLF-PHC for player i.

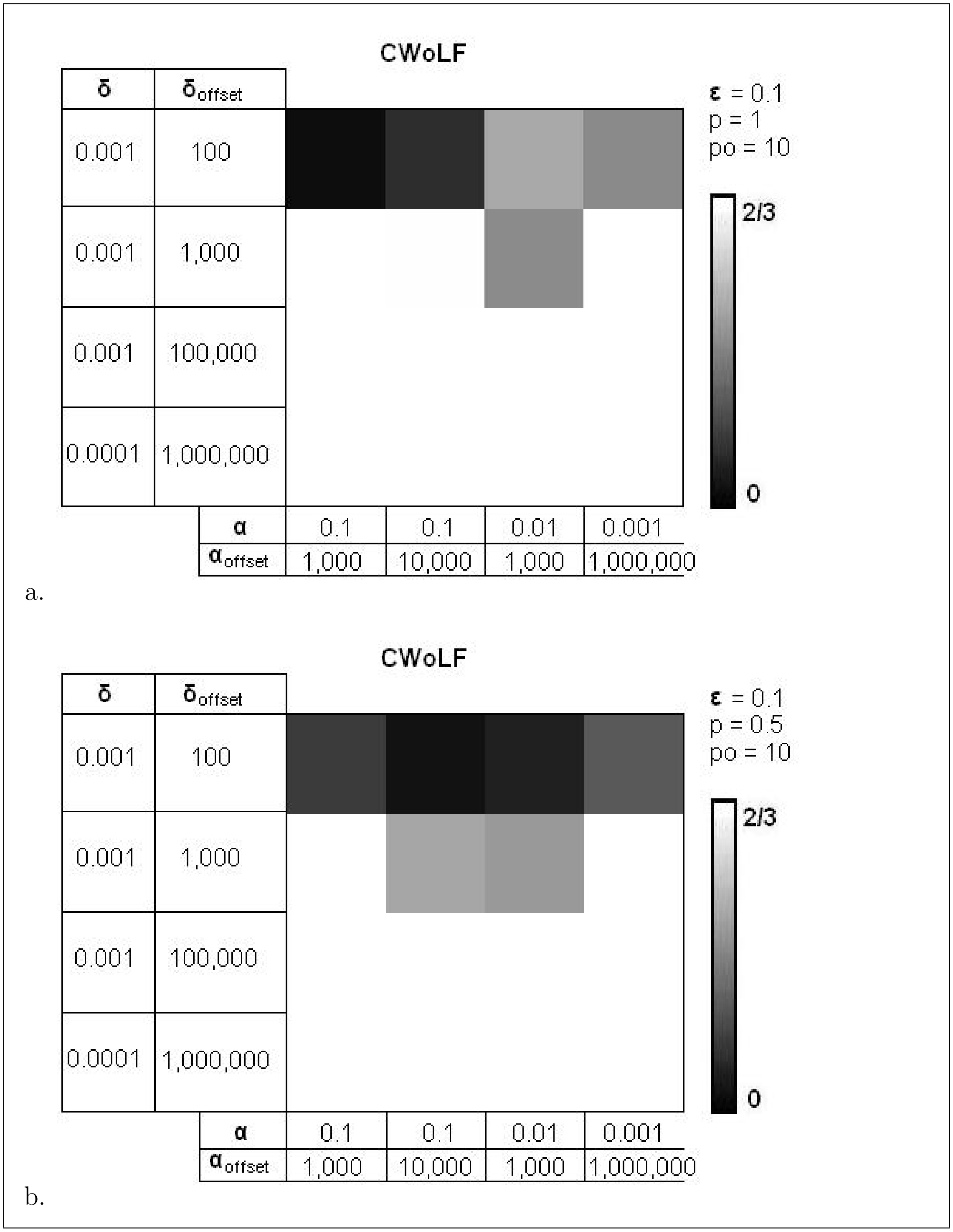

Figure B.3: (a) Results from 16 simulations using CWoLF-PHC using $p = 1$ and $p_{\text{offset}} = 10$. (b) Results from 16 simulations using CWoLF-PHC using $p = 0.5$ and $p_{\text{offset}} = 10$. The value plotted is the largest distance from 1/3 for any action's probability in either agent's policy after 900 million iterations.

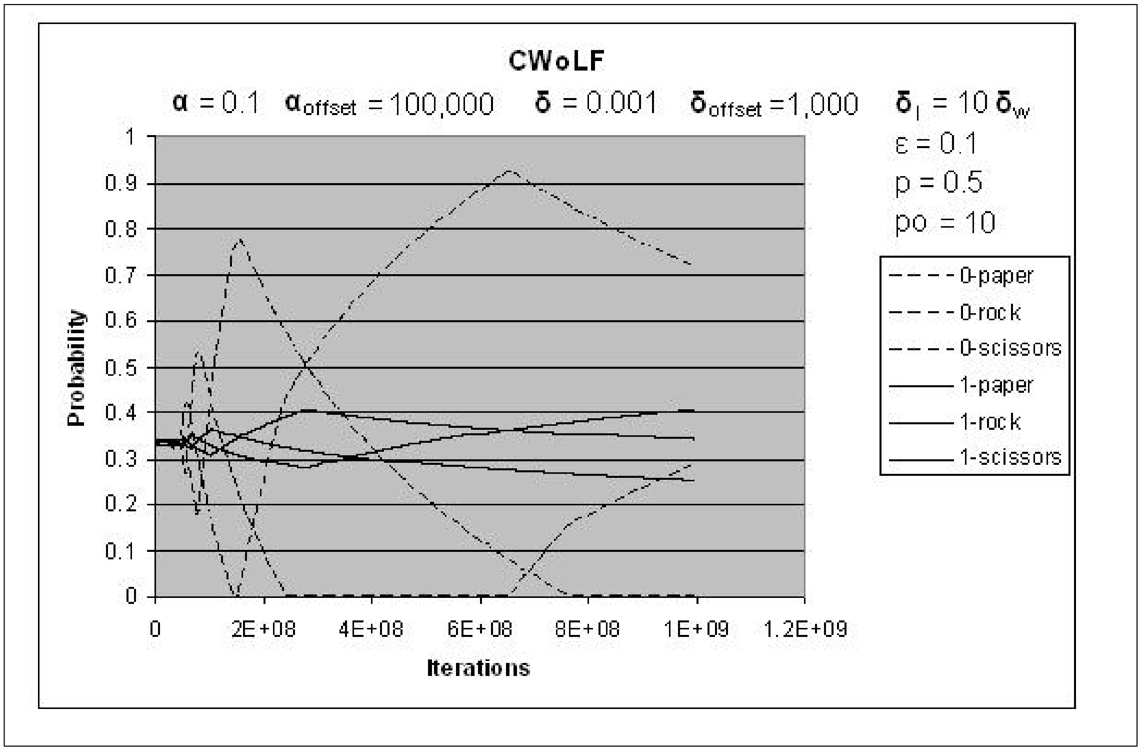

Figure B.4: Simulation of CWoLF in self-play in Shapley's game.

Agent 1 plays paper most often, while agent 2 plays rock most often. Agent 2 will realize that it should start playing scissors more and will eventually play it most often. Until agent 2 plays scissors most often, agent 1 will keep increasing the frequency of playing paper. Once agent 2 does play scissors with the greatest frequency, agent 1 starts changing its policy to play rock more. Agent 1 will have been increasing the frequency of paper during this whole time, and will receive a reward of -1 more than any other reward when playing paper. This helps the agent to decide that it is losing, and therefore use the larger delta value. The penalty received helps WoLF-PHC know that it has overlearned.

The RPWoLF mechanism subtracts a given amount from the reward if the action chosen happens to be a_{gfa}. This will help RPWoLF to determine more quickly that it has overlearned, and should allow the modification to determine that it is losing faster.

The algorithm for this modification is given in Table B.5. In our simulations, the function $F(r)$ equals $r - p$, where p is a real-valued parameter. Step 2c uses $F(r)$ if it identifies that the action played during this iteration is the most probable action without the highest Q-value. In other words, if the action chosen has the highest Q-value or if there exists an action played more frequently that does not have the highest Q-value, then the unmodified reward is used.

Figure B.5a and Figure B.5b show the results from the 16 simulations listed in Table 3.2 for two different values of p. As with CWoLF-PHC, values for p around 1 seem like they should help convergence. This algorithm obviously works better than the previous three, as can be seen with the bottom rows indicating that the agents did not diverge.

Figure B.6 shows two simulations using RPWoLF. These two graphs are different from the others in that they do not begin on the left at the first iteration. This is to show what the agent's policies are actually showing. Figure B.6a shows both agents' policies, and they seem to play nearly the same policy. Figure B.6b shows a simulation where the agents did not keep as close to playing random. Only one agent is shown, but the agent's policies seem to be out of phase by about half a period. This is likely due to the effect of the edge of the probability simplex. Notice that the policies in the simulations do not diverge, but they do not converge either. The policies tend cycle with a constant amplitude.

Since the rewards are being modified, two sets of Q-values should be used as in CWoLF. Simulations using two sets of Q-values diverged similar to WoLF-PHC. As was discussed in Section B.3, if two sets of Q-values are not used then the algorithm will not produce the same indifference points. This means RPWoLF will probably not converge to Nash equilibria based on indifference points.

B.5 Policy Penalizing WoLF-PHC (PWoLF-PHC)

RPWoLF-PHC modifies the rewards that the Q-learning layer receives. As discussed in Section B.3, this can cause the algorithm to fail to converge to the Nash equilibrium in certain games. Instead of modifying the rewards as in Section B.4, PWoLF-PHC modifies the policy. The PWoLF mechanism moves the policy away from the pure strategy associated with a_{gfa} after every iteration. The amount the policy is moved away from a_{gfa} is η times the amount moved towards the action with the highest Q-value. One way to think of this mechanism is that it assumes that the agent overlearned, so it tries to backtrack

```
1. Let $\alpha, \delta_l > \delta_w$ be learning rates. Let F(r) be a given function. Initialize,

       $Q(s,a) \leftarrow 0, \quad \pi(s,a) \leftarrow \frac{1}{|A_i|}, \quad C(s) \leftarrow 0.$

2. Repeat,

   (a) From state $s$ select action $a$ with probability $\pi(s,a)$, with suitable exploration

   (b) Observe reward $r_{\text{raw}}$ and next state $s'$.

   (c) Modify reward value to be used,

       $r = \begin{cases} r_{\text{raw}} & \text{if } a = \arg\max_{a'} Q(s,a') \\ r_{\text{raw}} & \text{if } a \neq \arg\max_{a'} \pi(s,a') \\ F(r_{\text{raw}}) & \text{otherwise.} \end{cases}$

   (d) Update Q-learning Layer,

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

   (e) Update estimate of average policy, $\bar{\pi}$,

       $C(s) \leftarrow C(s) + 1$
       $\forall a' \in A_i \quad \bar{\pi}(s,a') \leftarrow \bar{\pi}(s,a') + \frac{1}{C(s)}(\pi(s,a') - \bar{\pi}(s,a')).$

   (f) Step $\pi$ closer to the optimal policy w.r.t. Q. Same as PHC (2c), but with

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

Table B.5: Reward Penalizing WoLF-PHC algorithm for player i.

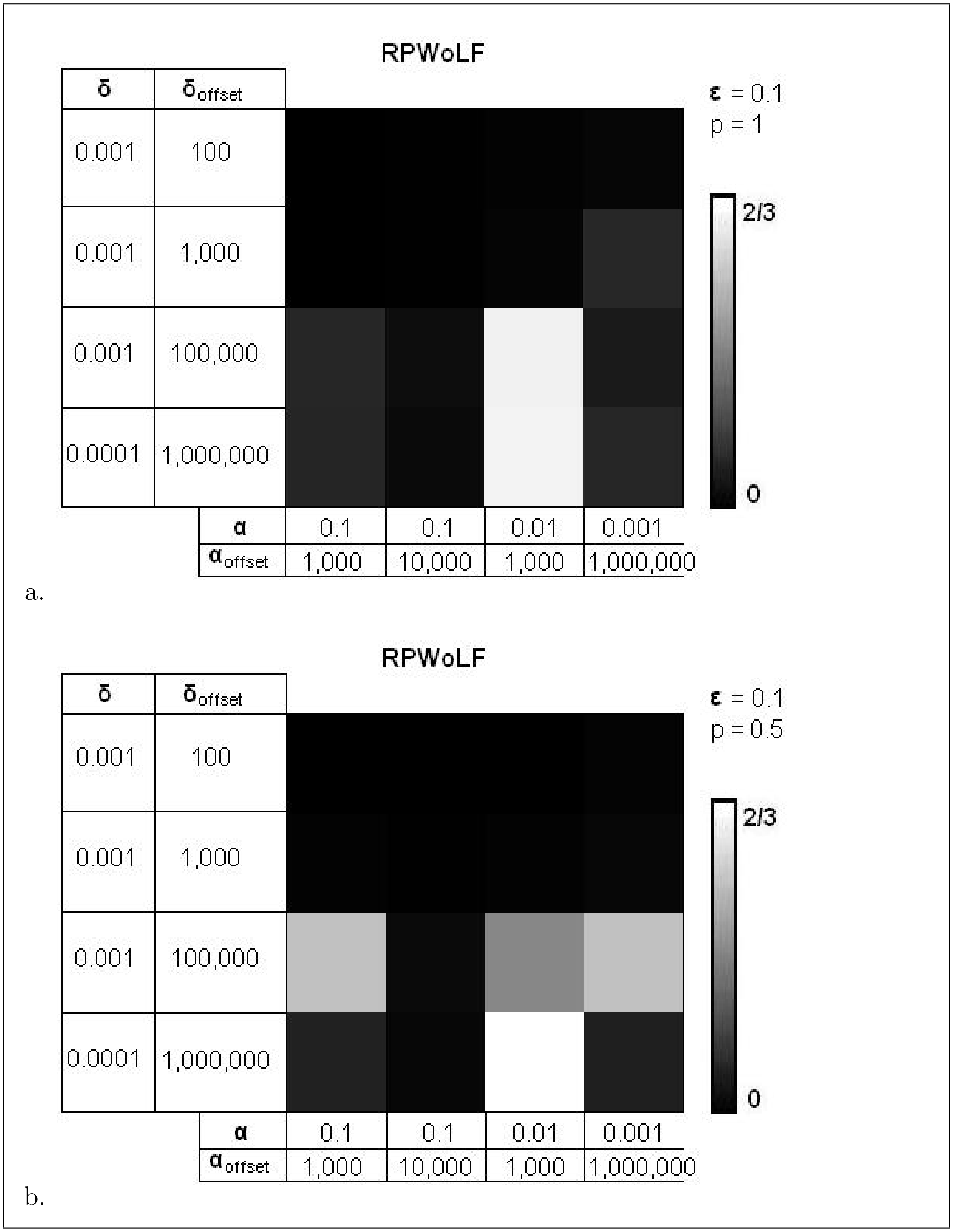

Figure B.5: (a) Results from 16 simulations using RPWoLF-PHC using $p = 1$. (b) Results from 16 simulations using RPWoLF-PHC using $p = 0.5$. The value plotted is the largest distance from 1/3 for any action's probability in either agent's policy after 900 million iterations.

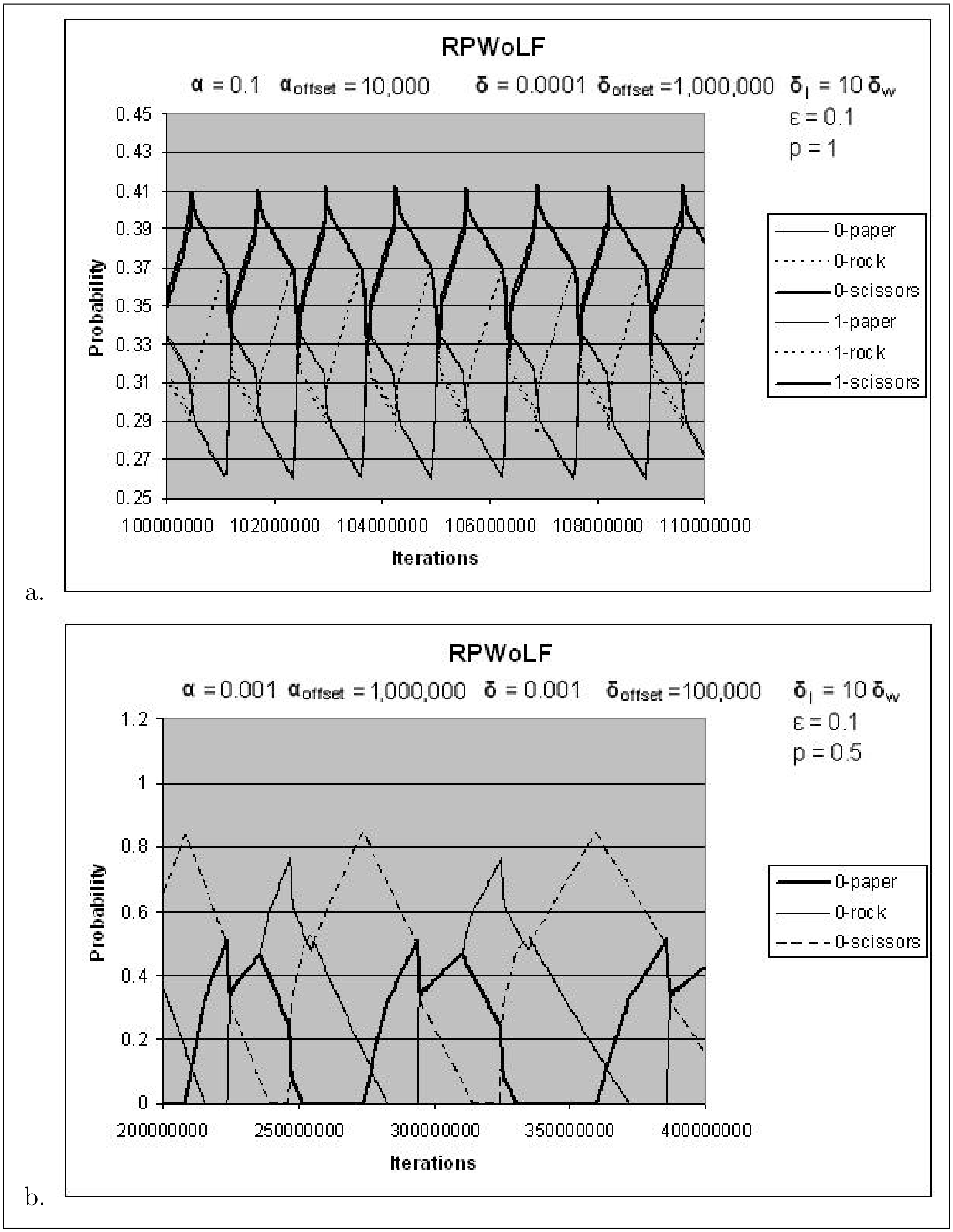

Figure B.6: Two simulations of RPWoLF-PHC in self-play in Shapley's game.

the policy a little as it learns new information.

The implementation for PWoLF-PHC is given in Table B.6. Step 2d in Table B.6 is where a_{gfa} is identified. Step 2f is where the PWoLF mechanism is implemented. A similar policy change mechanism to the WoLF-PHC policy change mechanism used in Step 2e is used in Step 2f, but Step 2f causes the policy to move away from a specified action instead of towards it. The policy change mechanism used by PHC and WoLF-PHC only allows the policy to change in as many ways as there are actions. PWoLF-PHC can move a policy in $n(n-1)$ directions, where n is the number of actions. This is due to first moving in the direction of the pure strategy associated with the highest Q-value, and then moving a possibly different amount towards one of the other $n-1$ pure strategies. In games with two actions PWoLF-PHC acts as WoLF-PHC with a larger delta.

Figure B.5a and Figure B.5b show the results from the 16 simulations listed in Table 3.2 for two different values of η. Values that were used for η are 0.5 and 1. These two figures do not seem to improve much upon WoLF-PHC. There is a difference when different values of η are used, but it is small. When $\eta = 1$ divergence happens slightly slower than with $\eta = 0.5$

Figure B.6 shows one simulation using PWoLF-PHC. Since $\eta = 1$ and Shapley's game is a three-person game, the least chosen action does not change until the next higher decreases to the same probability.

Notice that using $\eta > 1$ will cause even strictly dominated actions (actions that are worse in all situations than another action) to increase in probability. A good range for η is therefore (0,1). Since simulations with larger values of η show slower divergence and $\eta = 1$ does not allow convergence, it seems logical that this modification is no better than WoLF-PHC.

1. Let $\alpha, \delta_l > \delta_w$ be learning rates. Let $\eta \in [0, 1]$. Initialize,

$$Q(s,a) \leftarrow 0, \quad \pi(s,a) \leftarrow \frac{1}{|A_i|}, \quad C(s) \leftarrow 0.$$

2. Repeat,

    (a) From state s select action a with probability $\pi(s,a)$, with suitable exploration

    (b) Observing reward r and next state s',

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

    (c) Update estimate of average policy, $\bar{\pi}$,

    $$C(s) \leftarrow C(s) + 1,$$
    $$\forall a' \in A_i \quad \bar{\pi}(s,a') \leftarrow \bar{\pi}(s,a') + \frac{1}{C(s)}(\pi(s,a') - \bar{\pi}(s,a')).$$

    (d) Find the greatest falling action,

    $$a_{\text{gfa}} = \arg \max_{\{a' | a' \neq \arg\max_{a''} Q(s,a'')\}} \pi(s,a')$$

    (e) Step π closer to the optimal policy w.r.t. Q. Same as PHC (2c), but with

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

    (f) Step π away from a_{gfa}

    $$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} -\eta\delta & \text{if } a = a_{\text{gfa}} \\ \frac{\eta\delta}{|A_i|-1} & \text{otherwise} \end{cases}$$

Table B.6: Policy Penalizing WoLF-PHC algorithm for player i.

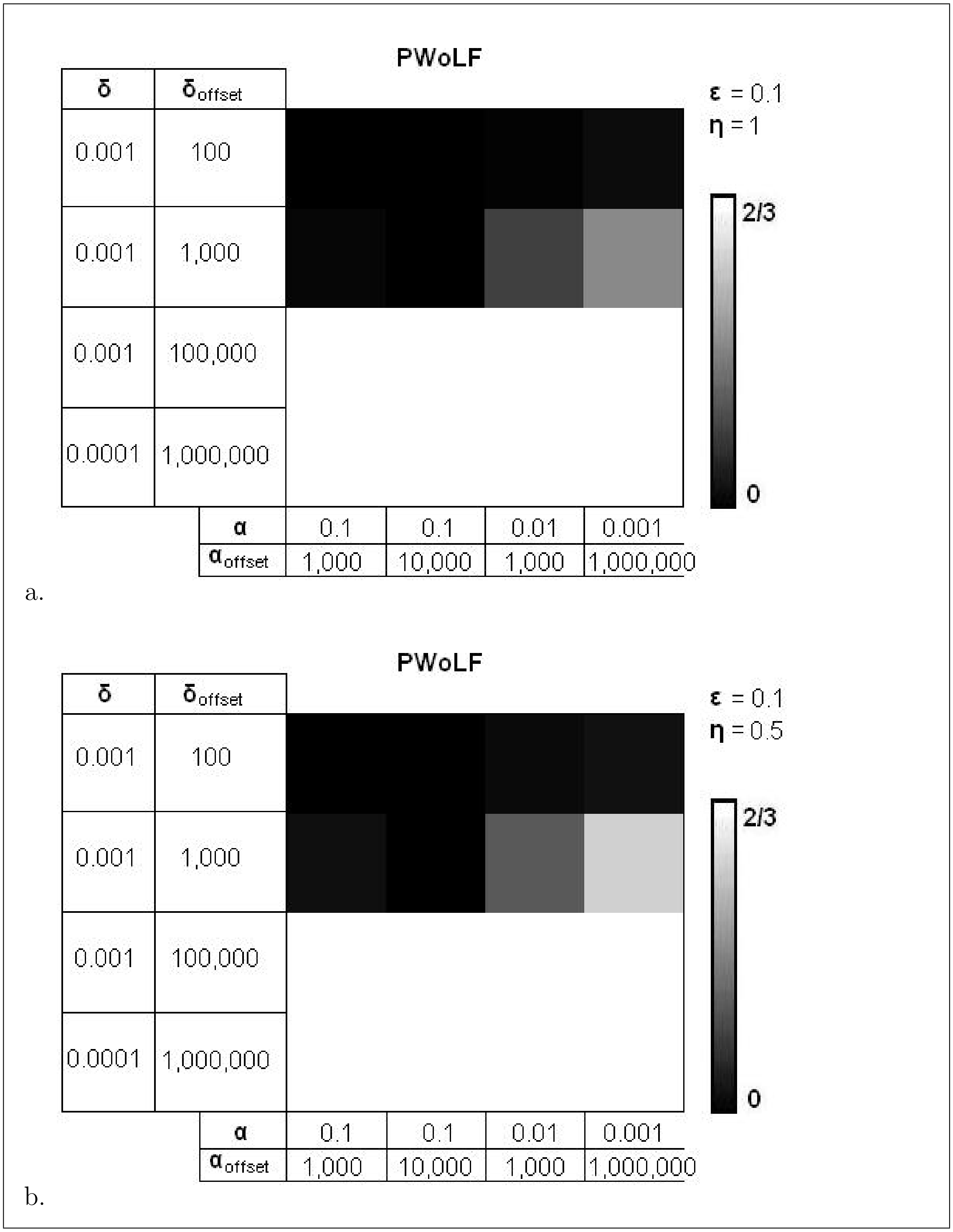

Figure B.7: (a) Results from 16 simulations using PWoLF-PHC using $\eta = 1$. (b) Results from 16 simulations using PWoLF-PHC using $\eta = 0.5$. The value plotted is the largest distance from 1/3 for any action's probability in either agent's policy after 900 million iterations.

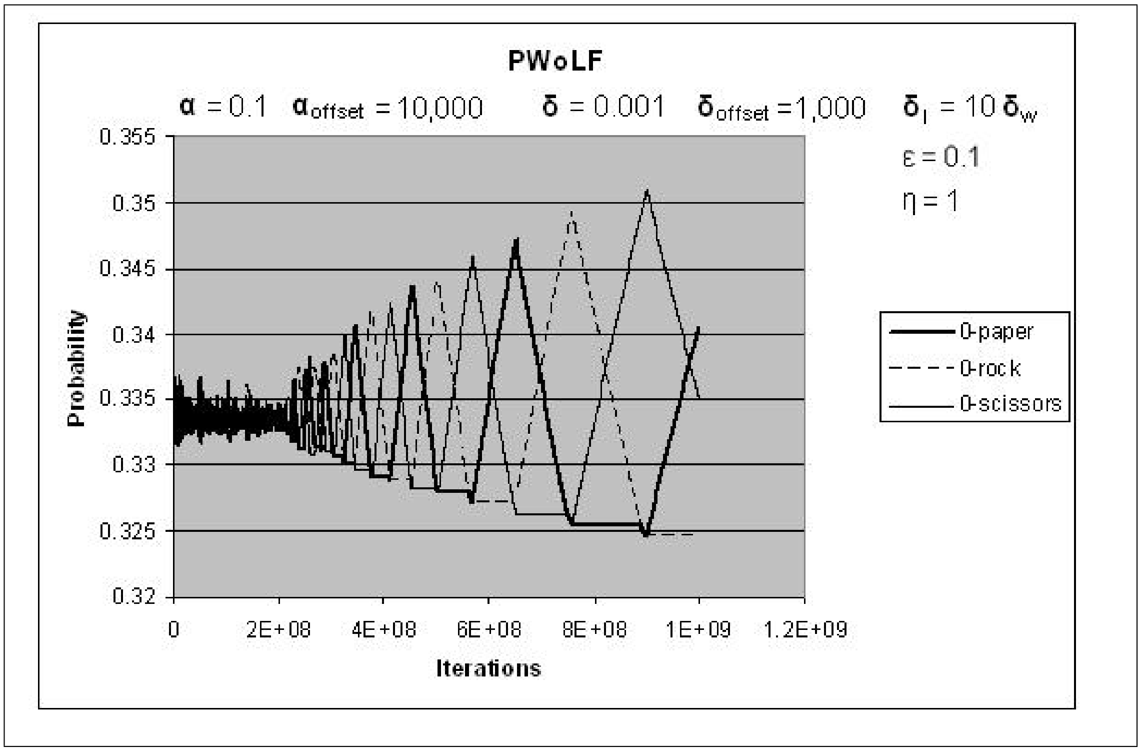

Figure B.8: Simulation of PWoLF-PHC in self-play in Shapley's game.

Appendix C

Parameters Used in PCWoLF-PHC Simulations

Due to the number of parameters that PCWoLF-PHC uses, the parameter values used in simulations are listed here rather than with the figures.

Figure 4.6a

Parameter	Value	Parameter	Value
α	.1	α_{offset}	10,000
δ_l	.01	δ_w	.001
δ_{offset}	1,000,000	η	.05
η_{offset}	5	ϵ	.98
ϵ_{offset}	20	τ_{best}	.5
τ_{streak}	2	τ_{shrink}	5
f_{stop}	2	f_{small}	.9
f_{middle}	1	f_{move}	.05
f_{shrink}	1.3	p_{stop}	3
γ	0		

Figure 4.6b and c

Parameter	Value	Parameter	Value
α	.001	α_{offset}	1,000,000
δ_l	.01	δ_w	.001
δ_{offset}	1,000,000	η	.05
η_{offset}	5	ϵ	.98
ϵ_{offset}	20	τ_{best}	.5
τ_{streak}	2	τ_{shrink}	5
f_{stop}	2	f_{small}	.9
f_{middle}	1	f_{move}	.05
f_{shrink}	1.4	p_{stop}	3
γ	0		

Figure 5.4

Parameter	Value	Parameter	Value
α	.1	α_{offset}	10,000
δ_l	.001	δ_w	.0001
δ_{offset}	100,000	η	.05
η_{offset}	5	ϵ	.98
ϵ_{offset}	20	τ_{best}	.05
τ_{streak}	2	τ_{shrink}	5
f_{stop}	2	f_{small}	.9
f_{middle}	1	f_{move}	.04
f_{shrink}	1.4	p_{stop}	3
γ	0		

Figure 5.5

Parameter	Value	Parameter	Value
α	.1	α_{offset}	10,000
δ_l	.001	δ_w	.0001
δ_{offset}	100,000	η	.05
η_{offset}	5	ϵ	.98
ϵ_{offset}	20	τ_{best}	.05
τ_{streak}	2	τ_{shrink}	5
f_{stop}	2	f_{small}	.9
f_{middle}	1	f_{move}	.05
f_{shrink}	1.4	p_{stop}	3
γ	0		

Figure 5.6

Parameter	Value	Parameter	Value
α	.001	α_{offset}	1,000,000
δ_l	.001	δ_w	.0001
δ_{offset}	100,000	η	.05
η_{offset}	5	ϵ	.98
ϵ_{offset}	20	τ_{best}	.5
τ_{streak}	2	τ_{shrink}	5
f_{stop}	2	f_{small}	.9
f_{middle}	1	f_{move}	.02
f_{shrink}	1.4	p_{stop}	3
γ	0		

Figure 5.7

Parameter	Value	Parameter	Value
α	.001	α_{offset}	1,000,000
δ_l	.001	δ_w	.0001
δ_{offset}	100,000	η	.05
η_{offset}	5	ϵ	.98
ϵ_{offset}	5	τ_{best}	.5
τ_{streak}	2	τ_{shrink}	5
f_{stop}	2	f_{small}	1
f_{middle}	1	f_{move}	.2
f_{shrink}	1.4	p_{stop}	3
γ	0		

Figure 5.8

Parameter	Value	Parameter	Value
α	.01	α_{offset}	10,000
δ_l	.01	δ_w	.001
δ_{offset}	1,000,000	η	.05
η_{offset}	5	ϵ	.98
ϵ_{offset}	20	τ_{best}	.5
τ_{streak}	2	τ_{shrink}	5
f_{stop}	2	f_{small}	.9
f_{middle}	1	f_{move}	.04
f_{shrink}	1.4	p_{stop}	3
γ	0		