Chapter 3
Learning in Two-Player Matrix Games

3.1 Matrix Games

In this chapter, we will examine the two-player stage game or the matrix game problem. Now, we have two players each learning how to play the game. In some cases they may be competing with each other, or they may be cooperating with other. In this section, we will introduce the class of game that we will investigate in this chapter. In fact, almost every child has played some version of these games. We will focus on three different games: matching pennies, rock-paper-scissors, and prisoners' dilemma. These are all called *matrix games* or *stage games* because there is no state transition involved. We will limit how far we delve into game theory and focus on the learning algorithms associated with these games. The idea is for the agents to play these games repetitively and learn their best strategy. In some cases one gets a pure strategy; in other words the agent will choose the same particular action all the time, and in some cases it is best to pick an action with a particular probability, which is known as a *mixed strategy*.

In the prisoners' dilemma game, two prisoners who committed a crime together are being interrogated by the police. Each prisoner has two choices; one choice is to cooperate with the police and defect on his accomplice, and the other is to cooperate with his accomplice and lie to the police. If both

Table 3.1 Examples of two-player matrix games.

(a) Matching Pennies	(b) Prisoners' Dilemma	(c) Rock-Paper-Scissors
$R_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix},$	$R_1 = \begin{bmatrix} 5 & 0 \\ 10 & 1 \end{bmatrix},$	$R_1 = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix},$
$R_2 = -R_1$	$R_2 = (R_1)^T$	$R_2 = -R_1$
NE in fully mixed strategies	NE in pure strategies	NE in fully mixed strategies

of them cooperate with each other and do not confess to the crime, then they will get just a few months in jail. If they both defect and cooperate with the police, then they will get a longer time in jail. However, if one of them defects and cooperates with the police and the other one cooperates with his accomplice and lies to the police, then the one who lied to the police and tried to cooperate with the accomplice will go to jail for a very long time. In Table 3.1, the payoff matrix for the game is shown. This matrix stipulates the rewards for player 1. In the matrix, the entries represent the rewards to the row player, and the first row represents cooperation with the accomplice and the second row represents defection and confession to the police. If the prisoners cooperate with each other and both of them pick the first row and column, then they only go to jail for a short time, a few months, and they get a good reward of 5. However, if row player defects and tells the truth to the police and the column player lies to the police and cooperates with his accomplice, the row player gets a big reward of 10 and goes free, whereas the column player would get a reward of 0 and be sent to jail for life. If they both defect and tell the truth to the police, then they each get a small reward of 1 and go to jail for a couple of years. If this was you, would you trust your criminal accomplice to cooperate with you because if he defects to the police and you lie to the police then you will go to jail for a very long time? Most rational people will confess to the police and limit the time that they may spend in jail. The choice of action to defect is known as the *Nash equilibrium* (NE). If a machine learning agent were to play this game repetitively, it should learn to play the action of *Defect* all the time, with 100% probability. This is known as a *pure* strategy game. A *pure* strategy means that one picks the same action all the time.

The next game we will define is the matching pennies game. In this game two children each hold a penny. They then independently choose to show either heads or tails. If they show two tails or two heads, then player 1 will win a reward of 1 and player 2 loses and gets a reward of -1. If they both show different sides of the coin, then player 2 wins. On any given play, one will win and one will lose. This is known as a *zero-sum matrix* game. When we say that it is a zero-sum game, we mean that one wins the same amount as the other loses. This game's *optimal* solution, or its NE, is the mixed strategy of choosing heads 50% of the time and choosing tails also 50% of the time. If player 2 always played heads, then quickly player 1 would realize that player 2 always plays heads and player 1 would also start to always play heads and would begin to win all the time. If player 2 always played heads, then we would say that player 2 was an irrational player. So clearly, each one of them should play either heads or tails 50% of the time to maximize their reward. This is known as a *mixed strategy* game; whereas in the prisoner's dilemma game the optimal strategy was always to defect 100% of the time and as such we refer to that as a *pure* strategy.

The next game of interest to us is the game of rock-paper-scissors. This game is well known to most children. The idea is to display your hand as either a rock (clenched fist), scissors, or as a flat piece of paper. Then, paper *covers* (beats) rock, rock *breaks* (beats) scissors, and scissors *cuts* (beats) paper. If both players display the same entity, then it is a tie. This game is a mixed strategy zero-sum game. The obvious solution is to randomly play each action, rock, paper, or scissors with a 33.3% probability. The only difference to this game is that we now have three actions instead of two.

More formally, a matrix game (strategic game) [1, 2] can be described as a tuple $(n, A_{1,\ldots,n}, R_{1,\ldots,n})$, where n is the agents' number, A_i is the discrete space of agent i's available actions, and R_i is the payoff function that agent i receives. In matrix games, the objective of agents is to find pure or mixed strategies that maximize their payoffs. A pure strategy is the strategy that chooses actions deterministically, whereas a mixed strategy is the strategy that chooses actions based on a probability distribution over the agent's available actions. The NE in the rock-paper-scissors game and the matching pennies game are mixed strategies that execute actions with equal probability [3].

The player i's reward function R_i is determined by all players' joint action from joint action space $A_1 \times \cdots \times A_n$. In a matrix game, each player tries to maximize its own reward based on the player's strategy. A player's strategy in a matrix game is a probability distribution over the player's action set. To evaluate a player's strategy, we introduce the following concept of NE:

Definition 3.1 *A Nash equilibrium in a matrix game is a collection of all players' strategies* $(\pi_1^*, \ldots, \pi_n^*)$ *such that*

$$V_i(\pi_1^*, \ldots, \pi_i^*, \ldots, \pi_n^*) \geq V_i(\pi_1^*, \ldots, \pi_i, \ldots, \pi_n^*), \tag{3.1}$$

$$\forall \pi_i \in \Pi_i, i = 1, \ldots, n \tag{3.2}$$

where $V_i(\cdot)$ *is player i's value function which is player i's expected reward given all players' strategies, and* π_i *is any strategy of player i from the strategy space* Π_i.

In other words, an NE is a collection of strategies for all players such that no player can do better by changing its own strategy given that other players continue playing their NE strategies [4]. We define $Q_i(a_1, \ldots, a_n)$ as the received reward of player i given players' joint action $a_1, \ldots, a_n$, and $\pi_i(a_i)$ $(i = 1, \ldots, n)$ as the probability of player i choosing action a_i. Then the NE defined in (3.1) becomes

$$\begin{aligned} &\sum_{a_1, \ldots, a_n \in A_1 \times \cdots \times A_n} Q_i(a_1, \ldots, a_n)\pi_1^*(a_1) \cdots \pi_i^*(a_i) \cdots \pi_n^*(a_n) \geq \\ &\sum_{a_1, \ldots, a_n \in A_1 \times \cdots \times A_n} Q_i(a_1, \ldots, a_n)\pi_1^*(a_1) \cdots \pi_i(a_i) \cdots \pi_n^*(a_n), \\ &\forall \pi_i \in \Pi_i, i = 1, \cdots, n \end{aligned} \tag{3.3}$$

where $\pi_i^*(a_i)$ is the probability of player i choosing action a_i under the player i's NE strategy π_i^*.

We provide the following definitions regarding matrix games:

Definition 3.2 *A Nash equilibrium is called a* ***strict*** *Nash equilibrium if (3.1) is strict [5].*

Definition 3.3 *If the probability of any action from the action set is greater than 0, then the player's strategy is called a* ***fully mixed strategy****.*

Definition 3.4 *If the player selects one action with probability 1 and other actions with probability 0, then the player's strategy is called a* ***pure strategy****.*

Definition 3.5 *A Nash equilibrium is called a* ***strict Nash equilibrium in pure strategies*** *if each player's equilibrium action is better than all its other actions, given the other players' actions [6].*

3.2 Nash Equilibria in Two-Player Matrix Games

For a two-player matrix game, we can set up a matrix with each element containing a reward for each joint action pair. Then the reward function R_i for player $i (i = 1, 2)$ becomes a matrix.

A two-player matrix game is called a *zero-sum game* if the two players are fully competitive. In this way, we have $R_1 = -R_2$. A zero-sum game has a unique NE in the sense of the expected reward. This means that, although each player may have multiple NE strategies in a zero-sum game, the value of the expected reward V_i under these NE strategies will be the same. A *general-sum matrix game* refers to all types of matrix games. In a general-sum matrix game, the NE is no longer unique and the game might have multiple NEs.

For a two-player matrix game, we define $\pi_i = (\pi_i(a_1), \dots, \pi_i(a_{m_i}))$ as the set of all probability distributions over player i's action set $A_i (i = 1, 2)$. Then V_i becomes

$$V_i = \pi_1 R_i \pi_2^T \tag{3.4}$$

An NE for a two-player matrix game is the strategy pair (π_1^*, π_2^*) for two players such that, for $i = 1, 2$,

$$V_i(\pi_i^*, \pi_{-i}^*) \geq V_i(\pi_i, \pi_{-i}^*), \forall \pi_i \in PD(A_i) \tag{3.5}$$

where $-i$ denotes any other player than player i, and $PD(A_i)$ is the set of all probability distributions over player i's action set A_i.

Given that each player has two actions in the game, we can define a two-player two-action general-sum game as

$$R_1 = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}, \; R_2 = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \tag{3.6}$$

where r_{lf} and c_{lf} denote the reward to the row player (player 1) and the reward to the column player (player 2), respectively. The row player chooses action $l \in \{1, 2\}$ and the column player chooses action $f \in \{1, 2\}$. Based on Definition 3.2 and (3.5), the pure strategies l and f are called a *strict NE in pure strategies* if

$$r_{lf} > r_{-lf}, c_{lf} > c_{l-f} \qquad \text{for } l, f \in \{1, 2\} \tag{3.7}$$

where $-l$ and $-f$ denote any row other than row l and any column other than column f, respectively.

3.3 Linear Programming in Two-Player Zero-Sum Matrix Games

One of the issues that arise in some of the machine learning algorithms is to solve for the NE. This is easier said than done. In this section, we will demonstrate how to compute the NE in competitive zero-sum games. In some of the algorithms to follow, a step in the algorithm will be to solve for the NE using linear programming or quadratic programming. To do this, we will be required to set up a constrained minimization/maximization problem that will be solved with the simplex method. The simplex method is well known in the linear programming community.

Finding the NE in a two-player zero-sum matrix game is equal to finding the minimax solution for the following equation [7]:

$$\max_{\pi_i \in PD(A_i)} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} R_i \pi_i(a_i) \tag{3.8}$$

where $\pi_i(a_i)$ denotes the probability distribution over player i's action a_i, and a_{-i} denotes any action from another player other than player i. According to (3.8), each player tries to maximize the reward in the worst case scenario against its opponent. To find the solution for (3.8), one can use linear programming.

Assume we have a 2×2 zero-sum matrix game given as

$$R_1 = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}, \; R_2 = -R_1 \tag{3.9}$$

where R_1 is player 1's reward matrix and R_2 is player 2's reward matrix. We define p_j $(j = 1, 2)$ as the probability distribution over player 1's jth action and q_j as the probability distribution over player 2's jth action.

Then the linear program for player 1 is

$$\text{Find } (p_1, p_2) \text{ to maximize } V_1$$

subject to

$$r_{11}p_1 + r_{21}p_2 \geq V_1 \tag{3.10}$$

$$r_{12}p_1 + r_{22}p_2 \geq V_1 \tag{3.11}$$

$$p_1 + p_2 = 1 \tag{3.12}$$

$$p_j \geq 0, \qquad j = 1, 2 \tag{3.13}$$

The linear program for player 2 is

$$\text{Find } (q_1, q_2) \text{ to maximize } V_2$$

subject to

$$-r_{11}q_1 - r_{12}q_2 \geq V_2 \tag{3.14}$$

$$-r_{21}q_1 - r_{22}q_2 \geq V_2 \tag{3.15}$$

$$q_1 + q_2 = 1 \tag{3.16}$$

$$q_j \geq 0, \qquad j = 1, 2 \tag{3.17}$$

To solve the above linear programming problem, one can use the simplex method to find the optimal points geometrically. We provide three 2×2 zero-sum games below.

Example 3.1 We take the matching pennies game, for example. The reward matrix for player 1 is

$$R_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{3.18}$$

Since $p_2 = 1 - p_1$, the linear program for player 1 becomes

$$\text{Player 1: find } p_1 \text{ to maximize } V_1$$

subject to

$$2p_1 - 1 \geq V_1 \tag{3.19}$$

$$-2p_1 + 1 \geq V_1 \tag{3.20}$$

$$0 \leq p_1 \leq 1 \tag{3.21}$$

We use the simplex method to find the solution geometrically. Figure 3-1 shows the plot of p_1 over V_1 where the gray area satisfies the constraints (3.19)–(3.21). From the plot, the maximum value of V_1 within the gray area is 0 when $p_1 = 0.5$. Therefore, $p_1 = 0.5$ is the Nash equilibrium strategy for player 1. Similarly, we can use the simplex method to find the Nash equilibrium strategy for player 2. After solving (3.14)–(3.17), we can find that the maximum value of V_2 is 0 when $q_1 = 0.5$. Then this game has a Nash equilibrium ($p_1 = 0.5,\ q_1 = 0.5$), which is a fully mixed strategy Nash equilibrium.

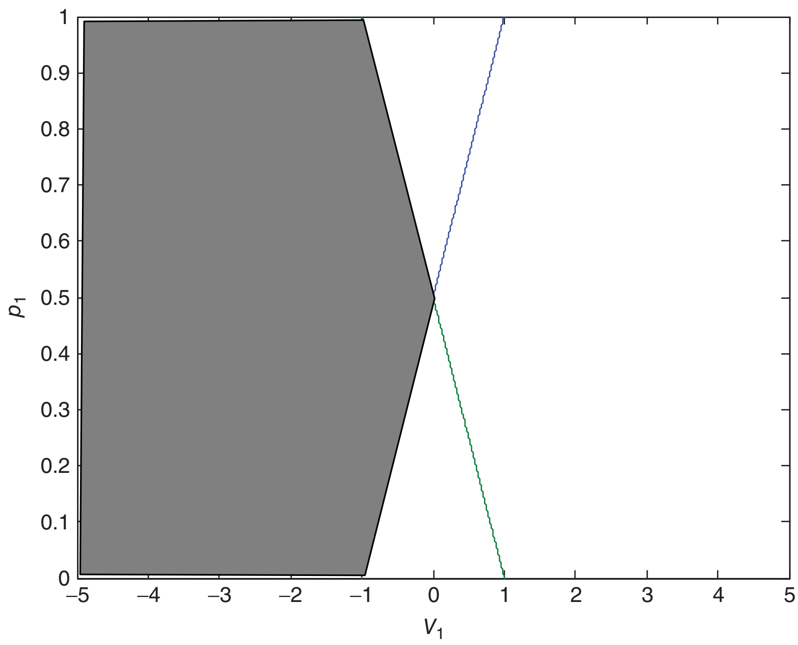

Fig. 3-1. Simplex method for player 1 in the matching pennies game. Reproduced from [8], © X. Lu.

Example 3.2 We change the reward r_{12} from -1 in (3.18) to 2 and call this game as the revised version of the matching pennies game. The reward matrix for player 1 becomes

$$R_1 = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} \tag{3.22}$$

The linear program for player 1 is

Player 1: find p_1 to maximize V_1

subject to

$$2p_1 - 1 \geq V_1 \tag{3.23}$$

$$p_1 + 1 \geq V_1 \tag{3.24}$$

$$0 \leq p_1 \leq 1 \tag{3.25}$$

From the plot in Fig. 3-2, we can find that the maximum value of V_1 in the gray area is 1 when $p_1 = 1$. Similarly, we can find the maximum value of

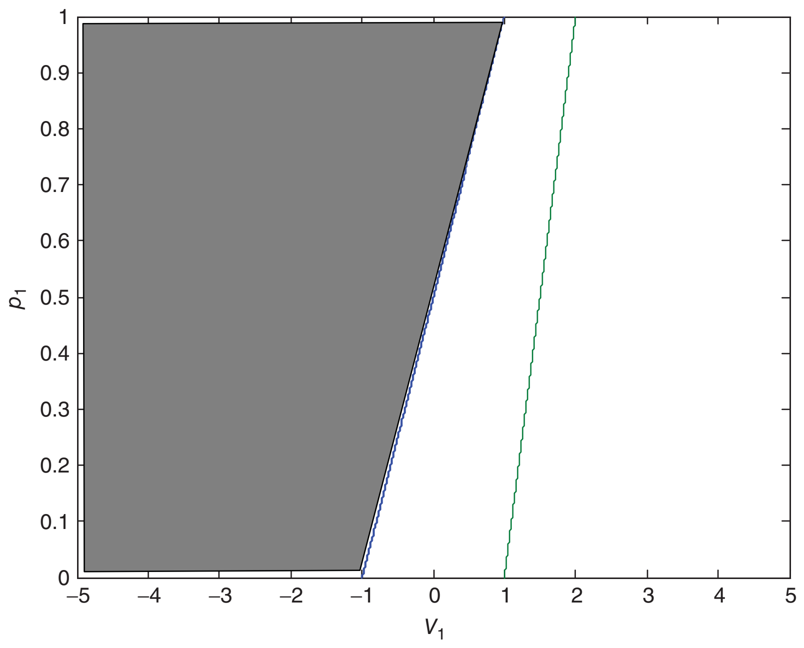

Fig. 3-2. Simplex method for player 1 in the revised matching pennies game. Reproduced from [8], © X. Lu.

$V_2 = -1$ when $q_1 = 1$. Therefore, this game has a Nash equilibrium ($p_1 = 1$, $q_1 = 1$), which is a pure strategy Nash equilibrium.

Example 3.3 We now consider the following zero-sum matrix game:

$$R_1 = \begin{bmatrix} r_{11} & 2 \\ 3 & -1 \end{bmatrix}, \; R_2 = -R_1 \tag{3.26}$$

where $r_{11} \in \mathfrak{R}$. Based on different values of r_{11}, we want to find the Nash equilibrium strategies (p_1, q_1). The linear program for each player becomes

$$\text{Player 1: Find } p_1 \text{ to maximize } V_1$$

subject to

$$(r_{11} - 3)p_1 + 3 \geq V_1 \tag{3.27}$$

$$3p_1 - 1 \geq V_1 \tag{3.28}$$

$$0 \leq p_1 \leq 1 \tag{3.29}$$

$$\text{Player 2: Find } q_1 \text{ to maximize } V_2$$

subject to

$$(2 - r_{11})q_1 - 2 \geq V_2 \tag{3.30}$$

$$-4q_1 + 1 \geq V_2 \tag{3.31}$$

$$0 \leq q_1 \leq 1 \tag{3.32}$$

We use the simplex method to find the Nash equilibria for the players with a varying r_{11}. When $r_{11} > 2$, we find that the Nash equilibrium is in pure strategies ($p_1^* = 1, q_1^* = 0$). When $r_{11} < 2$, we find that the Nash equilibrium is in fully mixed strategies ($p_1^* = 4/(6 - r_{11}), q_1^* = 3/(6 - r_{11})$). For $r_{11} = 2$, we plot the players' strategies over their value functions in Fig. 3-3. From the plot we find that player 1's Nash equilibrium strategy is $p_1 = 1$, and player 2's Nash equilibrium strategy is $q_1 \in [0, 0.75]$, which is a set of strategies. Therefore, at $r_{11} = 2$, we have multiple Nash equilibria which are $p_1 = 1, q_1 \in [0, 0.75]$. We also plot the Nash equilibria (p_1, q_1) over r_{11} in Fig. 3-4.

3.4 The Learning Algorithms

In this section, we will present several algorithms that have gained popularity within the field of machine learning. We will focus on the algorithms that have been used for learning how to choose the *optimal* actions when agents are playing matrix games. Once again, these algorithms will look like gradient descent (ascent) algorithms. We will discuss their strengths and weaknesses. In particular, we are going to look at the gradient ascent (GA) algorithm and its related version the infinitesimal gradient ascent (IGA) algorithm and the policy hill climbing (PHC) algorithm and the variable learning rate version called the *win or learn fast-policy hill climbing* (WoLF-PHC) algorithm [3]. We will then examine the linear reward-inaction (L_{RI}) and the lagging anchor algorithm. Finally, we will discuss the advantages of the L_{RI} lagging anchor algorithm. There are a number of versions of these algorithms in the literature, but they tend to be minor variations of the ones being discussed here. Of course, one could argue that all learning algorithms are minor variations of the stochastic approximation technique.

3.5 Gradient Ascent Algorithm

One of the fundamental algorithms associated with learning in matrix games is the GA algorithm and its related formulation called the *IGA* algorithm. This algorithm is used in relatively simple two-action/two-player general-sum

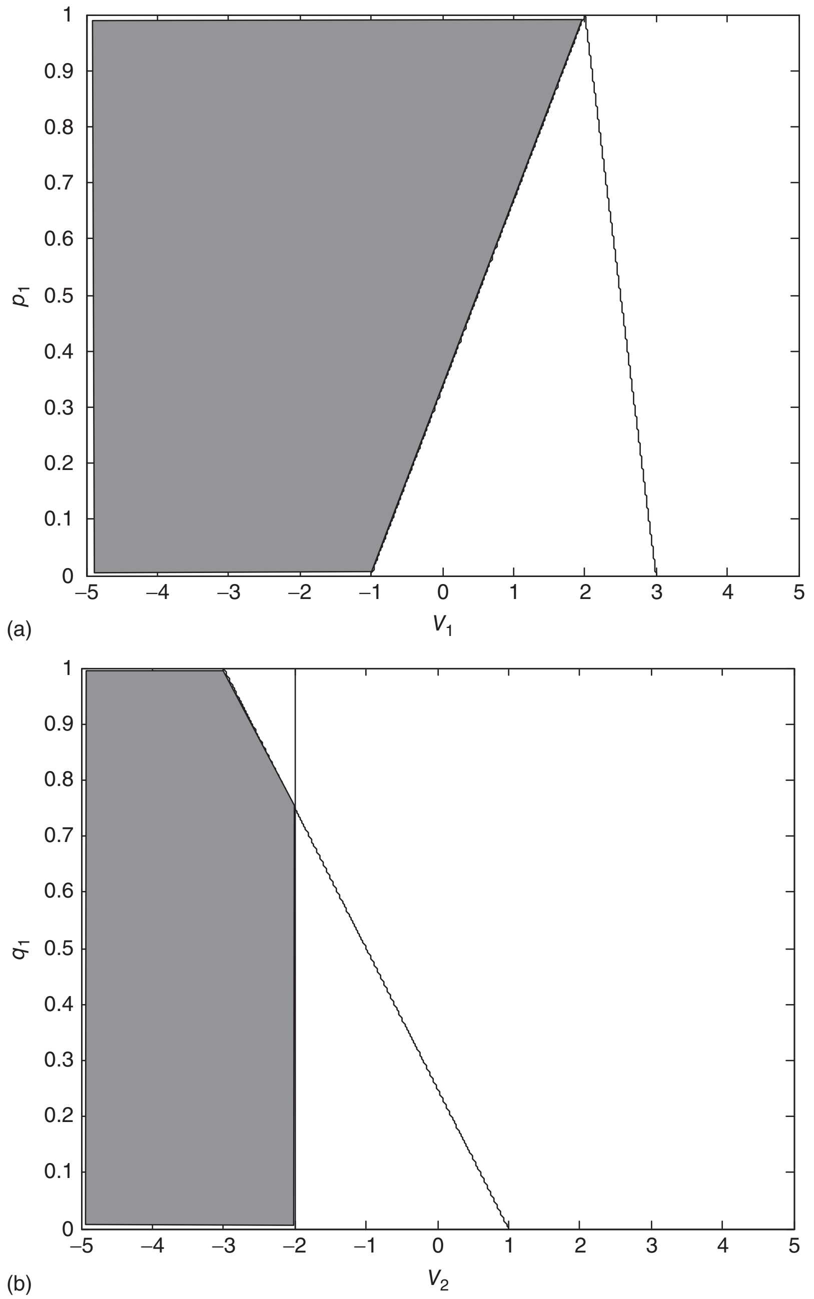

Fig. 3-3. Simplex method at $r_{11} = 2$ in Example 3.3. (a) Simplex method for player 1 at $r_{11} = 2$. (b) Simplex method for player 2 at $r_{11} = 2$. Reproduced from [8], © X. Lu.

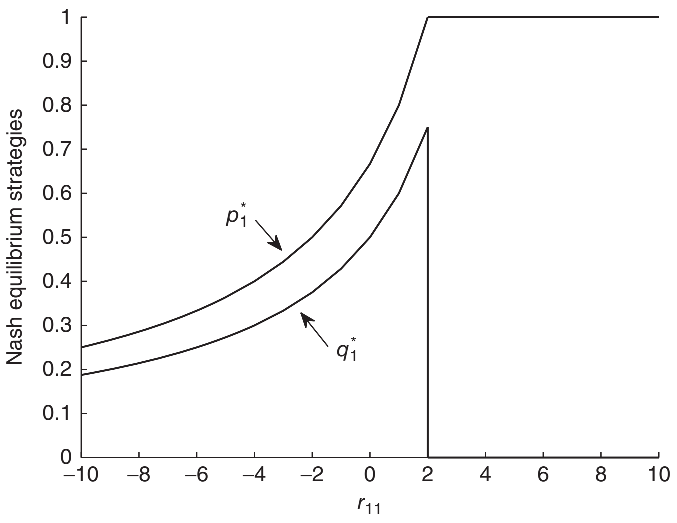

Fig. 3-4. Players' NE strategies versus r_{11}. Reproduced from [8], © X. Lu.

games. Theoretically, this algorithm will fail to converge. It can be shown that by introducing a variable learning rate that tends to zero as $\lim_{t\to\infty}\eta \to 0$, the GA algorithm will converge.

We will examine the GA algorithm presented by Singh et al. [9]. We examine the case of a 2×2 matrix game as two payoff matrices, one for the row player and one for the column player. The matrices are

$$R_r = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \tag{3.33}$$

and

$$R_c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \tag{3.34}$$

Then, if the row player chooses action 1 and the column player chooses action 2, then the reward to player 1 (the row player) is r_{12} and the reward to player 2 (the column player) is c_{12}. This is a two-action two-player game and we are assuming the existence of a mixed strategy, although the algorithm can be used for pure strategy games as well. In a mixed strategy game, the probability that the row player chooses action 1 is $P\{a_r = 1\} = \alpha$ and, therefore, the probability that the row player chooses action 2 must be given by $P\{a_r = 2\} = 1 - \alpha$. Similarly, for player 2 (the column player), the probability that player 2 chooses action 1 is given by $P\{a_c = 1\} = \beta$ and, therefore, the probability of choosing action 2 is $P\{a_c = 1\} = 1 - \beta$. The strategy of the matrix game is completely defined by the joint strategy $\pi(\alpha, \beta)$, where α and β are constrained to remain within the unit square. We define the expected payoff to each player as $V_r(\alpha, \beta)$ and $V_c(\alpha, \beta)$. We can write the expected payoffs as

$$V_r(\alpha, \beta) = \alpha\beta r_{11} + \alpha(1-\beta)r_{12} + (1-\alpha)\beta r_{21} \tag{3.35}$$

$$+(1-\alpha)(1-\beta)r_{22} \tag{3.36}$$

$$= u_r\alpha\beta + \alpha(r_{12} - r_{22}) + \beta(r_{21} - r_{22}) + r_{22} \tag{3.37}$$

$$V_c(\alpha, \beta) = \alpha\beta c_{11} + \alpha(1-\beta)c_{12} + (1-\alpha)\beta c_{21} \tag{3.38}$$

$$+(1-\alpha)(1-\beta)c_{22} \tag{3.39}$$

$$= u_c\alpha\beta + \alpha(c_{12} - c_{22}) + \beta(c_{21} - c_{22}) + c_{22} \tag{3.40}$$

where

$$u_r = r_{11} - r_{12} - r_{21} + r_{22} \tag{3.41}$$

$$u_c = c_{11} - c_{12} - c_{21} + c_{22} \tag{3.42}$$

We can now compute the gradient of the payoff function with respect to the strategy as

$$\frac{\partial V_r(\alpha, \beta)}{\partial \alpha} = \beta u_r + (r_{12} - r_{22}) \tag{3.43}$$

$$\frac{\partial V_c(\alpha, \beta)}{\partial \beta} = \alpha u_c + (c_{21} - c_{22}) \tag{3.44}$$

The GA algorithm then becomes

$$\alpha_{k+1} = \alpha_k + \eta \frac{\partial V_r(\alpha_k, \beta_k)}{\partial \alpha_k} \tag{3.45}$$

$$\beta_{k+1} = \beta_k + \eta \frac{\partial V_c(\alpha_k, \beta_k)}{\partial \beta_k} \tag{3.46}$$

Theorem 3.1 If both players follow infinitesmal gradient ascent (IGA), where $\eta \to 0$, then their strategies will converge to a Nash equilibrium, or the average payoffs over time will converge in the limit to the expected payoffs of a Nash equilibrium.

The first algorithm we will try is the GA algorithm. We will play the mixed strategy games of matching pennies. To implement the GA learning algorithm for the matching pennies game, one needs to know the payoff matrix in

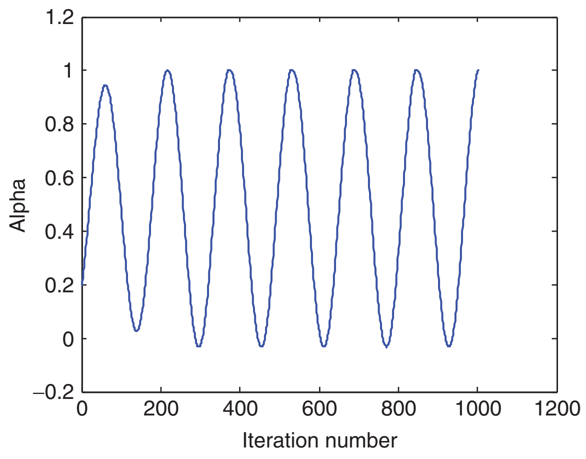

Fig. 3-5. GA in matching pennies game.

advance. One can see from Fig. 3-5 that the strategy oscillates between 0 and 1. If we try to implement the IGA algorithm, one runs into the difficulty of trying to choose an appropriate rate of convergence of the step size to zero. This is not a practical algorithm to use. Therefore, the GA algorithm does not work particularly well; it oscillates and one can show this theoretically [3].

3.6 WoLF-IGA Algorithm

The WoLF-IGA algorithm was introduced by Bowling and Veloso [3] for two-player two-action matrix games. As a GA learning algorithm, the WoLF-IGA algorithm allows the player to update its strategy based on the current gradient and a variable learning rate. The value of the learning rate is smaller when the player is winning, and it is larger when the player is losing. The term p_1 is the probability of player 1 choosing the first action. Then, $1 - p_1$ is the probability of player 1 choosing the second action. Accordingly, q_1 is the probability of player 2 choosing the first action and $1 - q_1$ is the probability of player 2 choosing the second action. The updating rules of the WoLF-IGA algorithm are as follows:

$$p_1(k+1) = p_1(k) + \eta\alpha_1(k)\frac{\partial V_1(p_1(k), q_1(k))}{\partial p_1} \tag{3.47}$$

$$q_1(k+1) = q_1(k) + \eta\alpha_2(k)\frac{\partial V_2(p_1(k), q_1(k))}{\partial q_1} \tag{3.48}$$

$$\alpha_1(k) = \begin{cases} \alpha_{\min}, & \text{if } V_1(p_1(k), q_1(k)) > V_1(p_1^*, q_1(k)) \\ \alpha_{\max}, & \text{otherwise} \end{cases}$$

$$\alpha_2(k) = \begin{cases} \alpha_{\min}, & \text{if } V_2(p_1(k), q_1(k)) > V_2(p_1(k), q_1^*) \\ \alpha_{\max}, & \text{otherwise} \end{cases}$$

where η is the step size, $\alpha_i (i = 1, 2)$ is the learning rate for player $i(i = 1, 2)$, $V_i(p_1(k), q_1(k))$ is the expected reward of player i at time k given the current two players' strategy pair $(p_1(k), q_1(k))$, and (p_1^*, q_1^*) are equilibrium strategies for the players. In a two-player two-action matrix game, if each player uses the WoLF-IGA algorithm with $\alpha_{\max} > \alpha_{\min}$, the players' strategies converge to an NE as the step size $\eta \to 0$ [3].

This algorithm is a GA learning algorithm that can guarantee the convergence to an NE in fully mixed or pure strategies for two-player two-action general-sum matrix games. However, this algorithm is not a decentralized learning algorithm. It requires the knowledge of $V_1(p_1^*, q_1(k))$ and $V_2(p_1(k), q_1^*)$ in order to choose the learning parameters $\alpha_{\min}$ and $\alpha_{\max}$ accordingly. In order to obtain $V_1(p_1^*, q_1(k))$ and $V_2(p_1(k), q_1^*)$, we need to know each player's reward matrix and its opponent's strategy at time k; whereas in a decentralized learning algorithm, the agents would only have their own actions and reward at time k. Although a practical decentralized learning algorithm called a *WoLF-PHC method* was provided in Reference 3, there is no proof of convergence to NE strategies.

3.7 Policy Hill Climbing (PHC)

A more practical version of the gradient descent algorithm is the PHC algorithm. This algorithm is based on the Q-learning algorithm that we presented in Chapter 2. This is a rational algorithm that can estimate mixed strategies. The algorithm will converge to the *optimal* mixed strategies if the other players are not learning and are therefore playing *stationary* strategies.

The PHC algorithm is a simple practical algorithm that can learn mixed strategies. Hill climbing is performed by the PHC algorithm in the space of the mixed strategies. This algorithm was first proposed by Bowling and Veloso [3]. The PHC does not require much information as neither the recent actions executed by the agent nor the current strategy of its opponent is required to be known. The probability that the agent selects the highest valued actions is increased by a small learning rate $\delta \in (0,1]$. The algorithm is equivalent to the single-agent Q-learning when $\delta = 1$ as the policy moves to the greedy policy with probability 1. The PHC algorithm is rational and converges to the optimal solution when a fixed (stationary) strategy is followed by the other players. However, the PHC algorithm may not converge to a stationary policy if the other players are learning [3].

The convergence proof is the same as for Q-learning [10], which guarantees that the Q values will converge to the optimal Q^* with a suitable exploration policy [9]. However, when both players are learning, then the algorithm will not necessarily converge. The algorithm starts from the Q-learning algorithm and is given as

$$Q^j_{t+1}(a) = (1-\alpha)Q^j_t(a) + \alpha(r^j + \gamma \max_{a'} Q^j_t(a')) \tag{3.49}$$

$$\pi^j_{t+1}(a) = \pi^j_t(a) + \Delta_a \tag{3.50}$$

where

$$\Delta_a = \begin{cases} -\delta_a & \text{if } a \neq \text{argmax}_{a'} Q^j_t(a') \\ \sum_{a' \neq a} \delta_{a'} & \text{otherwise} \end{cases}$$

where $\delta_a = \min\left(\pi^j_t(a), \frac{\delta}{|A_j|-1}\right)$

The algorithm is given as,

Algorithm 3.1 Policy hill-climbing (PHC) algorithm for agent j:

```
Initialize:
learning rates $\alpha \in (0,1]$, $\delta \in (0,1]$
discount factor $\gamma \in (0,1)$
exploration rate $\epsilon$
$Q^j(a) \leftarrow 0$ and $\pi^j(a) \leftarrow \frac{1}{|A_j|}$
Repeat
(a) Select an action $a$ according to the strategy $\pi^j_t(a)$ with some exploration $\epsilon$.
(b) Observe the immediate reward $r^j$.
(c) Update $Q^j_{t+1}(a)$ using Eq. (3.49).
(d) Update the strategy $\pi^j_{t+1}(a)$ by using Eq. (3.50).
```

We will now run a simulation of the matching pennies games. To generate the simulation results illustrated in Fig. 3-6, we set the learning rate $\alpha = 1/(10 + 0.00001t)$, the exploration rate to $\epsilon = 0.5/(1 + 0.0001t)$, and $\delta = 0.0001$. We initialize the probability of player 1 choosing action 1, at 80%. One can see that the algorithm will oscillate about the NE as expected by the theory. In this case, both players are learning. For any practical application, this is a poor

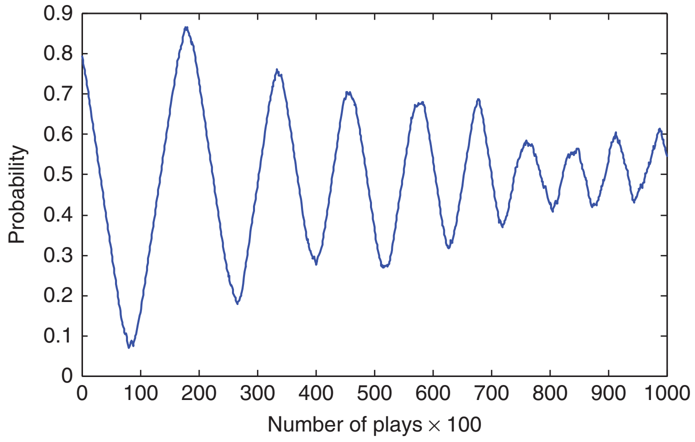

Fig. 3-6. PHC matching pennies game, player 1, probability of choosing action 1, heads.

result. Furthermore, it takes many iterations to converge about the 50% equilibrium point. Another issue with implementing this algorithm is choosing all the parameters. In a more complex game, this algorithm would not be practical to implement.

In the next case, we set the column player to always play heads, action 1, and we start the row player at 20% heads and 80% tails. Then the row player should learn to always play heads 100% of the time. As illustrated in Fig. 3-7, the probability of player 1 choosing heads increases and converges to a probability of 100%.

3.8 WoLF-PHC Algorithm

In Reference 3, the authors propose to use a variable learning rule as

$$\alpha_{k+1} = \alpha_k + \eta l_k^r \frac{\partial V_r(\alpha_k, \beta_k)}{\partial \alpha_k} \tag{3.51}$$

$$\beta_{k+1} = \beta_k + \eta l_k^c \frac{\partial V_c(\alpha_k, \beta_k)}{\partial \beta_k} \tag{3.52}$$

where the term l is a variable learning rate given by $l \in [l_{\min}, l_{\max}] > 0$.

The method for adjusting the learning rate l is referred to as the *WoLF* approach. The idea is when one is winning the game to adjust the learning rate to learn slowly and be cautious, and when losing or doing poorly to learn quickly. The next step is to determine when the agent is doing well or doing

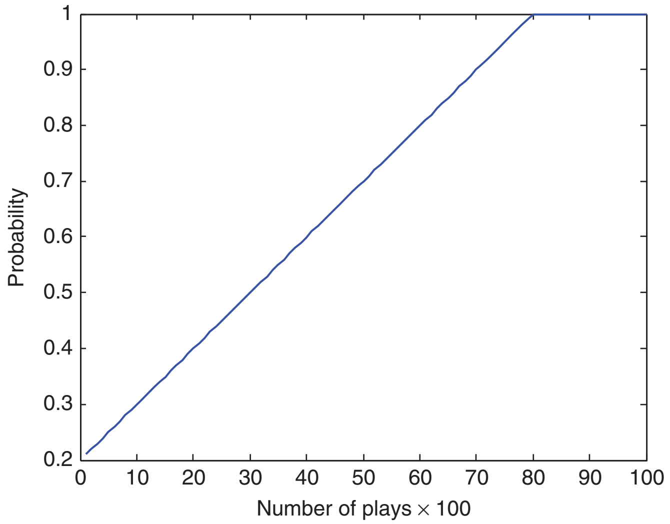

Fig. 3-7. PHC matching pennies game, player 1, probability of choosing action 1, heads when player 2 always chooses heads.

poorly in playing the game. The conceptual idea is for the agent to choose an NE and compare the expected reward it would receive to the NE. If the reward it would receive is greater than the NE, then it is winning and will learn slowly and cautiously. Otherwise, it is losing and it should learn fast; the agent does want to be losing.

The two players each select an NE of their choice independently; they do not need to choose the same equilibrium point. If there are multiple NE points in the game, then the agents could pick different points; that is perfectly acceptable because each NE point will have the same value. Therefore, player 1 may choose NE point α^e and player 2 may choose NE point β^e, and the learning rates are chosen as

$$l_k^r = \begin{cases} l_{\min} & \text{if } V_r(\alpha_k, \beta_k) > V_r(\alpha^e, \beta_k) \quad \text{Winning} \\ l_{\max} & \text{otherwise} \quad \text{Losing} \end{cases}$$

$$l_k^c = \begin{cases} l_{\min} & \text{if } V_c(\alpha_k, \beta_k) > V_c(\alpha_k, \beta^e) \quad \text{Winning} \\ l_{\max} & \text{otherwise} \quad \text{losing} \end{cases}$$

When we combine the variable learning rate with the IGA algorithm, we refer to it as the *WoLF-IGA algorithm*. Although this is not a practical algorithm to implement, it does have good theoretical properties as defined by the following theorem.

Theorem 3.2 If in a two-action iterated general-sum game both players follow the WoLF-IGA algorithm (with $l_{\max} > l_{\min}$), then their strategies will converge to a Nash Equilibrium.

It is interesting to note that winning is defined as the expected reward of the current strategy being greater than the expected reward of the current player's NE strategy and the other player's current strategy.

The difficulty with the WoLF-IGA algorithm is the amount of information that the player must have. The player needs to know its own payoff matrix, the other player's strategy, and its own NE. Of course, if one knows its own payoff matrix, then it will also know its NE point or points. That is a lot of information for the player to know, and as such this is not a practical algorithm to implement.

The WoLF-PHC algorithm is an extension of the PHC algorithm [3]. This algorithm uses the mechanism of win-or-learn-fast (WoLF) so that the PHC algorithm converges to an NE in self-play. The algorithm has two different learning rates, δ_w when the algorithm is winning and δ_l when it is losing. The difference between the average strategy and the current strategy is used as a criterion to decide when the algorithm wins or loses. The learning rate δ_l is larger than the learning rate δ_w. As such, when a player is losing, it learns faster than when winning. This causes the player to adapt quickly to the changes in the strategies of the other player when it is doing more poorly than expected and learns cautiously when it is doing better than expected. This also gives the other player the time to adapt to the player's strategy changes. The WoLF-PHC algorithm exhibits the property of convergence as it makes the player converge to one of its NEs. This algorithm is also a rational learning algorithm because it makes the player converge to its optimal strategy when its opponent plays a stationary strategy. These properties permit the WoLF-PHC algorithm to be widely applied to a variety of stochastic games [3, 11–13]. The recursive Q-learning of a learning agent j is given as

$$Q^j_{t+1}(a) = (1-\alpha)Q^j_t(a) + \alpha(r^j + \gamma \max_{a'} Q^j_t(a')) \tag{3.53}$$

The WoLF-PHC algorithm updates the strategy of the agent j by equation 3.54, whereas Algorithm 2.1 describes the complete formal definition of the WoLF-PHC algorithm for a learning agent j:

$$\pi^j_{t+1}(a) = \pi^j_t(a) + \Delta_a \tag{3.54}$$

where

$$\Delta_a = \begin{cases} -\delta_a & \text{if } a \neq \text{argmax}_{a'} Q_t^j(a') \\ \sum\limits_{a' \neq a} \delta_{a'} & \text{otherwise} \end{cases} \qquad \delta_a = \min\left(\pi_t^j(a), \frac{\delta}{|A_j| - 1}\right)$$

$$\delta = \begin{cases} \delta_w & \text{if } \sum\limits_{a'} \pi_t(a') Q_{t+1}^j(a') > \\ & \quad \sum\limits_{a'} \bar{\pi}_{t+1}(a') Q_{t+1}^j(a') \\ \delta_l & \text{otherwise} \end{cases}$$

$$\bar{\pi}_{t+1}^j(a') = \bar{\pi}_t^j(a') + \frac{1}{C_{t+1}}(\pi_t^j(a') - \bar{\pi}_t^j(a')) \quad \forall a' \in A_j$$

$$C_{t+1} = C_t + 1$$

Algorithm 3.2 The win-or-learn-fast policy hill climbing (WoLF-PHC) algorithm for agent j

```
Initialize:
learning rates $\alpha \in (0,1]$, $\delta_w \in (0,1]$ and $\delta_l > \delta_w$
discount factor $\gamma \in (0,1)$
exploration rate $\epsilon$
$Q^j(a) \leftarrow 0$    and    $\pi^j(a) \leftarrow \frac{1}{|A_j|}$
$C(s) \leftarrow 0$
Repeat
(a) Select an action $a$ according to the strategy $\pi_t^j(a)$ with some exploration $\epsilon$.
(b) Observe the immediate reward $r^j$.
(c) Update $Q_{t+1}^j(a)$ using Eq. (3.53).
(d) Update the strategy $\pi_{t+1}^j(a)$ by using Eq. (3.54).
```

We simulate the WoLF-PHC algorithm for the matching pennies game. We set the learning parameter $\alpha = 1/(10 + 0.00001t)$ and $\delta_w = 1/(20000 + j)$ and $\delta_l = 2\delta_w$. The strategy for player 1 is initially set to $\pi_r = [0.2 \;\; 0.8]$ and the strategy for player 2 to $\pi = [0.5 \;\; 0.5]$. The results are shown in Fig. 3-8.

3.9 Decentralized Learning in Matrix Games

Decentralized learning means that there is no central learning strategy for all of the agents. Instead, each agent learns its own strategy. Decentralized learning

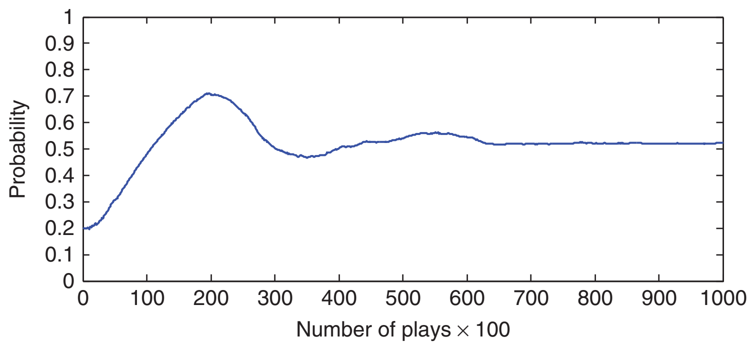

Fig. 3-8. WoLF-PHC matching pennies game, player 1, probability of choosing action 1.

algorithms can be used by players to learn their NEs in games with incomplete information [14, 15]. When an agent has "incomplete information," it means that the agent knows neither its own reward function, nor the other players' strategies, nor the other players' reward functions. The agent only knows its own action and the received reward at each time step. The main challenge for designing a decentralized learning algorithm with incomplete information is to prove that the players' strategies converge to an NE.

There are a number of multiagent learning algorithms proposed in the literature that can be used for two-player matrix games. Lakshmivarahan and Narendra [14] presented a linear reward–inaction approach that can guarantee the convergence to a NE under the assumption that the game only has strict NEs in pure strategies. The linear reward–penalty approach, introduced in Reference 15, can guarantee that the expected value of players' strategies converges to an NE in fully mixed strategies with the proper choice of parameters. Bowling and Veloso proposed a WoLF-IGA approach that can guarantee the convergence to an NE for two-player two-action matrix games and the NE can be in fully mixed strategies or in pure strategies. However, the WoLF-IGA approach is not a completely decentralized learning algorithm because the player has to know its opponent's strategy at each time step. Dahl [16, 17] proposed a lagging anchor model approach that can guarantee the convergence to an NE in fully mixed strategies. But the lagging anchor algorithm is not a decentralized learning algorithm because each player has to know its reward matrix.

We evaluate the learning automata algorithm L_{R-I} [14] and L_{R-P} [15], the GA algorithm WoLF-IGA [3], and the lagging anchor algorithm [16]. We then propose the new L_{R-I} lagging anchor algorithm. The L_{R-I} lagging anchor algorithm is a combination of learning automata and GA learning. It is a completely decentralized algorithm and, therefore, each agent only needs to know its own action and its own reward at each time step. We prove the convergence of the L_{R-I} lagging anchor algorithm to NEs in two-player two-action general-sum matrix games. Furthermore, the NE can be in games with pure or fully mixed strategies. We then simulate three matrix games to test the performance of the L_{R-I} lagging anchor learning algorithm.

We first review the multiagent learning algorithms in matrix games based on the learning automata scheme and the GA schemes. In Section 3.14, we introduce the new L_{R-I} lagging anchor algorithm and provide the proof of convergence to NEs in two-player two-action general-sum matrix games. Simulations of three matrix games are also illustrated in Section 3.14 to show the convergence of our proposed L_{R-I} lagging anchor algorithm.

3.10 Learning Automata

Learning in a two-player matrix game can be expressed as the process of each player updating its strategy according to the received reward from the environment. A learning scheme is used for each player to update its own strategy toward a NE based on the information from the environment. In order to address the limitations of the previously published multiagent learning algorithms for matrix games, we divide these learning algorithms into two groups. One group is based on learning automata [18], and another group is based on GA learning [9].

Learning automation is a learning unit for adaptive decision making in an unknown environment [18]. The objective of learning automation is to learn the optimal action or strategy by updating its action probability distribution based on the environment response. The learning automata approach is a completely decentralized learning algorithm because each learner only considers its action and the received reward from the environment and ignores any information from other agents such as the actions taken by other agents. The learning automation can be represented as a tuple (A, r, p, U), where $A = \{a_1, \dots, a_m\}$ is the player's action set, $r \in [0, 1]$ is the reinforcement signal, p is the probability distribution over the actions, and U is the learning algorithm to update p. There are two typical learning algorithms based on learning automata: the linear reward–inaction (L_{R-I}) algorithm and the linear reward–penalty (L_{R-P}) algorithm.

3.11 Linear Reward–Inaction Algorithm

The linear reward–inaction (L_{R-I}) algorithm for player $i (i = 1, \dots, n)$ is defined as follows:

$$\begin{aligned} p_c^i(k+1) &= p_c^i(k) + \eta r^i(k)(1 - p_c^i(k)) && \text{if } a_c \text{is the current action at } k \\ p_j^i(k+1) &= p_j^i(k) - \eta r^i(k) p_j^i(k) && \text{for all } a_j^i \neq a_c^i \end{aligned} \tag{3.55}$$

where k is the time step, the superscripts and subscripts on p denote different players and each player's different action, respectively, $0 < \eta < 1$ is the learning parameter, $r^i(k)$ is the response of the environment given player i's action a_c^i at k, and p_c^i is the probability distribution over player i's action $a_c^i (c = 1, \dots, m)$.

In a matrix game with n players, if each player uses the L_{R-I} algorithm, then the L_{R-I} algorithm guarantees the convergence to a NE under the assumption that the game only has strict NEs in pure strategies [14].

3.12 Linear Reward–Penalty Algorithm

The linear reward–penalty (L_{R-P}) algorithm for player i is defined as follows:

$$\begin{aligned} p_c^i(k+1) &= p_c^i(k) + \eta_1 r^i(k)[1 - p_c^i(k)] - \eta_2[1 - r^i(k)] p_c^i(k) \\ p_j^i(k+1) &= p_j^i(k) - \eta_1 r^i(k) p_j^i(k) + \eta_2[1 - r^i(k)] \left[\frac{1}{m-1} - p_j^i(k)\right] \text{(for all } a_j^i \neq a_c^i) \end{aligned} \tag{3.56}$$

where a_c^i is the current action the player i has taken, $0 < \eta_1, \eta_2 < 1$ are learning parameters, and m is the number of actions in the player's action set.

In a two-player zero-sum matrix game, if each player uses the L_{R-P} and chooses $\eta_2 < \eta_1$, then the expected value of the fully mixed strategies for both players can be made arbitrarily close to an NE [15]. This means that the L_{R-P} algorithm can guarantee the convergence to an NE in the sense of expected value but not the player's strategy itself.

3.13 The Lagging Anchor Algorithm

The lagging anchor algorithm for two-player zero-sum games was introduced by Dahl [16]. As a GA learning method, the lagging anchor algorithm updates the players' strategies according to the gradient. We denote player 1's strategy as a vector $\mathbf{v} = [p_1, p_2, \dots, p_{m_1}]^T$, which is the probability distribution over all the possible actions. Accordingly, player 2's strategy is denoted as a vector $\mathbf{w} = [q_1, q_2, \dots, q_{m_2}]^T$. The updating rules are listed as follows:

$$\begin{aligned} \mathbf{v}(k+1) &= \mathbf{v}(k) + \eta \mathbf{P}_{m_1} R_1 Y(k) + \eta\gamma(\bar{\mathbf{v}}(k) - \mathbf{v}(k)) \\ \bar{\mathbf{v}}(k) &= \bar{\mathbf{v}}(k) + \eta\gamma(\mathbf{v}(k) - \bar{\mathbf{v}}(k)) \\ \mathbf{w}(k+1) &= \mathbf{w}(k) + \eta \mathbf{P}_{m_2} R_2 X(k) + \eta\gamma(\bar{\mathbf{w}}(k) - \mathbf{w}(k)) \\ \bar{\mathbf{w}}(k) &= \bar{\mathbf{w}}(k) + \eta\gamma(\mathbf{w}(k) - \bar{\mathbf{w}}(k)) \end{aligned} \tag{3.57}$$

where η is the step size, $\gamma > 0$ is the anchor drawing factor, and $\mathbf{P}_{m_i} = \mathbf{I}_{m_i} - (1/m_i)\mathbf{1}_{m_i}\mathbf{1}_{m_i}^T$ is a matrix used to maintain the summation of the elements in the vector $\mathbf{v}$ or $\mathbf{w}$ to be 1. $Y(k)$ is a unit vector corresponding to the actions of player 2. If the m_ith action in player 2's action set is selected at time k, then the m_ith element in $Y(k)$ is set to 1 and the other elements in $Y(k)$ are zeros. Similarly, $X(k)$ is the unit vector corresponding to the actions of player 1, and R_1 and R_2 are the reward matrices for player 1 and 2, respectively. In (3.57), $\bar{\mathbf{v}}$ and $\bar{\mathbf{w}}$ are the anchor parameters for $\mathbf{v}$ and $\mathbf{w}$, respectively, which can be represented as the weighted average of the players' strategies. In a two-player zero-sum game with only NEs in fully mixed strategies, if each player uses the lagging anchor algorithm, then the players' strategies converge to an NE as the step size $\eta \to 0$ [17].

This algorithm guarantees the convergence to an NE in fully mixed strategies. However, the convergence to an NE in pure strategies has never been discussed. Furthermore, the lagging anchor algorithm in (3.57) requires full information of the player's reward matrices R_1 and R_2. Therefore, the lagging anchor algorithm is not a decentralized learning algorithm.

Table 3.2 compares these algorithms based on the allowable number of actions for each player, the convergence to pure strategies or fully mixed strategies, and the level of decentralization. From this table, only the WoLF-IGA algorithm can guarantee the convergence to both pure and mixed-strategy NE. But it is not a decentralized learning algorithm. Although the L_{R-I} algorithm and the L_{R-P} algorithm are decentralized learning algorithms, neither of them can guarantee the convergence to both pure and mixed-strategy NE. The L_{R-I} lagging anchor algorithm presented in the next section can guarantee convergence of both pure and mixed strategies to the NE as shown in Table 3.2.

Table 3.2 Comparison of learning algorithms in matrix games.

	Existing algorithms				Our proposed algorithm
Applicability	L_{R-I}	L_{R-P}	WoLF-IGA	Lagging anchor	L_{R-I} lagging anchor
Allowable actions	No limit	Two actions	Two actions	No limit	Two actions
Convergence	Pure NE	Fully mixed NE (expected value)	Both	Fully mixed NE	Both
Decentralized?	Yes	Yes	No	No	Yes

3.14 L_{R-I} Lagging Anchor Algorithm

In this section, we design an L_{R-I} lagging anchor algorithm which is a completely decentralized learning algorithm and can guarantee the convergence to NEs in both pure and fully mixed strategies. We take the L_{R-I} algorithm defined in (3.55) as the updating law of the player's strategy and add the lagging anchor term in (3.57). Then the L_{R-I} lagging anchor algorithm for player i is defined as follows:

$$\left.\begin{aligned} p_c^i(k+1) &= p_c^i(k) + \eta r^i(k)[1 - p_c^i(k)] + \eta[\bar{p}_c^i(k) - p_c^i(k)] \\ \bar{p}_c^i(k+1) &= \bar{p}_c^i(k) + \eta[p_c^i(k) - \bar{p}_c^i(k)] \end{aligned}\right\} \begin{array}{c}\text{if } a_c^i \text{ is the action} \\ \text{taken at time } k\end{array}$$

$$\left.\begin{aligned} p_j^i(k+1) &= p_j^i(k) - \eta r^i(k) p_j^i(k) + \eta[\bar{p}_j^i(k) - p_j^i(k)] \\ \bar{p}_j^i(k+1) &= \bar{p}_j^i(k) + \eta[p_j^i(k) - \bar{p}_j^i(k)] \end{aligned}\right\} \text{for all } a_j^i \neq a_c^i \tag{3.58}$$

where η is the step size and $(\bar{p}_c^i, \bar{p}_j^i)$ are the lagging parameters for (p_c^i, p_j^i). The idea behind the L_{R-I} lagging anchor algorithm is that we consider both the player's current strategy and the long-term average of the player's previous strategies at the same time. We expect that the player's current strategy and the long-term average will be drawn toward the equilibrium point during learning.

To analyze the above L_{R-I} lagging anchor algorithm, we use ordinary differential equations (ODEs). The behavior of the learning algorithm can be approximated by ODEs as the step size goes to zero. Thathachar and Sastry [19] provided the equivalent ODEs of the L_{R-I} algorithm in (3.55) as

$$\dot{p}_c^i = \sum_{j=1}^{m_i} p_c^i p_j^i (d_c^i - d_j^i) \tag{3.59}$$

where d_c^i is the expected reward given that player i is choosing action a_c^i and the other players are following their current strategies.

Combining the above ODEs of the L_{R-I} algorithm in (3.59) with the ODEs for the lagging anchor part of our algorithm, we can find the equivalent ODEs for our L_{R-I} lagging anchor algorithm, given as

$$\begin{aligned} \dot{p}_c^i &= \sum_{j=1}^{m_i} p_c^i p_j^i (d_c^i - d_j^i) + (\bar{p}_c^i - p_c^i) \\ \dot{\bar{p}}_c^i &= p_c^i - \bar{p}_c^i \end{aligned} \tag{3.60}$$

Based on our proposed L_{R-I} lagging anchor algorithm, we now present the following theorem:

Theorem 3.3 We consider a two-player two-action general-sum matrix game and assume the game only has a Nash equilibrium in fully mixed strategies or strict Nash equilibria in pure strategies. If both players follow the L_{R-I} lagging anchor algorithm, when the step size $\eta \to 0$, then the following is true regarding the asymptotic behavior of the algorithm:

- All Nash equilibria are asymptotically stable;
- Any equilibrium point which is not a Nash equilibrium is unstable.

Proof: Given a two-player two-action general-sum game defined in (3.6), we denote p_1 as the probability of player 1 taking its first action and q_1 as the probability of player 2 taking its first action. Then the L_{R-I} lagging anchor algorithm becomes

$$\begin{aligned}
\dot{p}_1 &= \sum_{j=1}^{2} p_1 p_j (d_1^1 - d_j^1) + (\bar{p}_1 - p_1)\\
\dot{\bar{p}}_1 &= p_1 - \bar{p}_1\\
\dot{q}_1 &= \sum_{j=1}^{2} q_1 q_j (d_1^2 - d_j^2) + (\bar{q}_1 - q_1)\\
\dot{\bar{q}}_1 &= q_1 - \bar{q}_1
\end{aligned} \tag{3.61}$$

where $d_1^1 = r_{11}q_1 + r_{12}(1-q_1)$, $d_2^1 = r_{21}q_1 + r_{22}(1-q_1)$, $d_1^2 = c_{11}p_1 + c_{21}(1-p_1)$, and $d_2^2 = c_{12}p_1 + c_{22}(1-p_1)$. Then (3.61) becomes

$$\begin{aligned}
\dot{p}_1 &= p_1(1-p_1)[u_1 q_1 + r_{12} - r_{22}] + (\bar{p}_1 - p_1)\\
\dot{\bar{p}}_1 &= p_1 - \bar{p}_1\\
\dot{q}_1 &= q_1(1-q_1)[u_2 p_1 + c_{21} - c_{22}] + (\bar{q}_1 - q_1)\\
\dot{\bar{q}}_1 &= q_1 - \bar{q}_1
\end{aligned} \tag{3.62}$$

where $u_1 = r_{11} - r_{12} - r_{21} + r_{22}$ and $u_2 = c_{11} - c_{12} - c_{21} + c_{22}$. If we let the right-hand side of the above equation equal to zero, we then get the equilibrium points of the above equations as $(p_1^*, q_1^*) = (0,0), (0,1), (1,0), (1,1), ((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$. To study the stability of the above learning dynamics, we use a linear approximation of the above equations around the equilibrium point $(p_1^*, q_1^*, p_1^*, q_1^*)$. Then the linearization matrix J is given as

$$J_{(p_1^*,q_1^*)} = \begin{bmatrix} (1-2p_1^*)(u_1 q_1^* + r_{12} - r_{22}) - 1 & 1 & p_1^*(1-p_1^*)u_1 & 0 \\ 1 & -1 & 0 & 0 \\ q_1^*(1-q_1^*)u_2 & 0 & (1-2q_1^*)(u_2 p_1^* + c_{21} - c_{22}) - 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{3.63}$$

If we substitute each of the equilibrium points $(0,0), (0,1), (1,0), (1,1)$ into (3.63), we get

$$J_{\text{pure}} = \begin{bmatrix} -e_1 - 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & -e_2 - 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{3.64}$$

where

$$e_1 = r_{22} - r_{12}, e_2 = c_{22} - c_{21} \quad \text{for } (0,0); \tag{3.65}$$

$$e_1 = r_{21} - r_{11}, e_2 = c_{21} - c_{22} \quad \text{for } (0,1); \tag{3.66}$$

$$e_1 = r_{12} - r_{22}, e_2 = c_{12} - c_{11} \quad \text{for } (1,0); \tag{3.67}$$

$$e_1 = r_{11} - r_{21}, e_2 = c_{11} - c_{12} \quad \text{for } (1,1) \tag{3.68}$$

The eigenvalues of the above matrix J_{pure} are $\lambda_{1,2} = 0.5[-(e_1+2) \pm \sqrt{e_1^2+4})]$ and $\lambda_{3,4} = 0.5[-(e_2+2) \pm \sqrt{e_2^2+4})]$. In order to obtain a stable equilibrium point, the real parts of the eigenvalues of J_{pure} must be negative. Therefore, the equilibrium point is asymptotically stable if

$$\begin{aligned} 0.5[-(e_{1,2}+2) \pm \sqrt{e_{1,2}^2+4})] < 0 \quad &\Rightarrow \\ e_{1,2} + 2 > \sqrt{e_{1,2}^2+4} \quad &\Rightarrow \\ e_{1,2} > 0 \end{aligned} \tag{3.69}$$

For the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$, the linearization matrix becomes

$$J_{\text{mixed}} = \begin{bmatrix} -1 & 1 & p_1^*(1 - p_1^*)u_1 & 0 \\ 1 & -1 & 0 & 0 \\ q_1^*(1 - q_1^*)u_2 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{3.70}$$

The characteristic equation of the above matrix is

$$\lambda^4 + 4\lambda^3 + (4 + e_3)\lambda^2 + 2e_3\lambda + e_3 = 0 \tag{3.71}$$

where $e_3 = -p_1^*(1 - p_1^*)q_1^*(1 - q_1^*)u_1u_2$. We set up the Routh table to analyze the locations of the roots in (3.71) as follows:

λ^4	1	$4 + e_3$	e_3
λ^3	4	$2c_3$	
λ^2	$4 + 0.5\,e_3$	e_3	
λ^1	$(e_3^2 + 4e_3)/(4 + 0.5e_3)$		
λ^0	e_3		

(3.72)

Based on the Routh–Hurwitz stability criterion, if (3.71) is stable, then all the coefficients of the equation must be positive and all the elements in the first column of the Routh table in (3.72) are positive. In order to meet the Routh–Hurwitz stability criterion, we must have $e_3 > 0$. Therefore, the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$ is asymptotically stable if

$$\begin{aligned} e_3 = -p_1^*(1 - p_1^*)q_1^*(1 - q_1^*)u_1u_2 > 0 \quad &\Rightarrow \\ u_1u_2 < 0 \end{aligned} \tag{3.73}$$

■

Case 3.1 Strict Nash equilibrium in pure strategies We first consider that the game only has strict Nash equilibrium in pure strategies. Without loss of generality, we assume that the Nash equilibrium in this case is both players' first actions. According to the definition of a strict Nash equilibrium in the inequality (3.7), if the Nash equilibrium strategies are both players' first actions, we can get

$$r_{11} > r_{21}, c_{11} > c_{12} \tag{3.74}$$

Since the Nash equilibrium in this case is the equilibrium point $(1, 1)$, we can get $e_1 = r_{11} - r_{21} > 0$ and $e_2 = c_{11} - c_{12} > 0$ based on (3.68) and (3.74). Therefore, the stability condition (3.69) is satisfied and the equilibrium point $(1, 1)$, which is the Nash equilibrium in this case, is asymptotically stable.

We now test the other equilibrium points. We first consider the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$. According to the stability condition (3.73), if this equilibrium point is stable, we must have $u_1u_2 < 0$. To be a valid inner point in the probability space (unit square), the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$ must satisfy

$$\begin{cases} 0 < (c_{22} - c_{21})/u_2 < 1 \\ 0 < (r_{22} - r_{12})/u_1 < 1 \end{cases} \tag{3.75}$$

If $u_1u_2 < 0$, we can get

$$\begin{cases} r_{11} > r_{21}, r_{22} > r_{12} \\ c_{11} < c_{12}, c_{22} < c_{21} \end{cases} \quad \text{if } u_1 > 0, u_2 < 0 \tag{3.76}$$

$$\begin{cases} r_{11} < r_{21}, r_{22} < r_{12} \\ c_{11} > c_{12}, c_{22} > c_{21} \end{cases} \quad \text{if } u_1 < 0, u_2 > 0 \tag{3.77}$$

However, the conditions in the inequalities (3.76) and (3.77) conflict with the inequalities in (3.74). Therefore, the inequality $u_1u_2 < 0$ will not hold and the equilibrium point $((c_{22} - c_{21})/u_2,(r_{22} - r_{12})/u_1)$ is unstable in Case 3.1.

For the equilibrium points $(0, 1)$ and $(1, 0)$, based on (3.66), (3.67), and (3.69), the stability conditions are $r_{21} > r_{11}, c_{21} > c_{22}$ for $(0, 1)$ and $r_{12} > r_{22}, c_{12} > c_{11}$ for $(1, 0)$. However, these stability conditions conflict with the inequalities $r_{11} > r_{21}, c_{11} > c_{12}$ in (3.74). Therefore, the equilibrium points $(0, 1)$ and $(1, 0)$ are unstable in Case 3.1.

For the equilibrium point $(0, 0)$, the stability condition is $r_{22} > r_{12}, c_{22} > c_{21}$ based on conditions (3.65) and (3.69). From the inequality (3.7), we can find that this stability condition also meets the requirement for a strict NE (both players' second actions) in inequality (3.7). Therefore, the equilibrium point $(0, 0)$ is stable only if it is also an NE point.

Thus, the NE point is asymptotically stable, while any equilibrium point which is not an NE is unstable.

Case 3.2 Nash equilibrium in fully mixed strategies We now consider that the game only has Nash equilibrium in fully mixed strategies. Singh et al. [9] showed that a Nash equilibrium in fully mixed strategies for a two-player two-action general-sum matrix game has the form

$$(p_1^{NE}, q_1^{NE}) = \left[\frac{c_{22} - c_{21}}{u_2}, \frac{r_{22} - r_{12}}{u_1}\right] \tag{3.78}$$

where (p_1^{NE}, q_1^{NE}) denotes the Nash equilibrium strategies over players' first actions which happens to be the equilibrium point of (3.62). According to the condition (3.73), the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$ is asymptotically stable if $u_1 u_2 < 0$. If we assume $u_1 u_2 > 0$, we can get

$$\begin{cases} 0 < (c_{22} - c_{21})/u_2 < 1 \\ 0 < (r_{22} - r_{12})/u_1 < 1 \end{cases}$$

$$\begin{cases} r_{11} > r_{21}, r_{22} > r_{12} \\ c_{11} > c_{12}, c_{22} > c_{21} \end{cases} \quad \text{if } u_1 > 0, u_2 > 0 \tag{3.79}$$

$$\begin{cases} r_{11} < r_{21}, r_{22} < r_{12} \\ c_{11} < c_{12}, c_{22} < c_{21} \end{cases} \quad \text{if } u_1 < 0, u_2 < 0 \tag{3.80}$$

According to the inequality (3.7), the above equations contain multiple NEs in pure strategies: $(p_1^{NE}, q_1^{NE}) = (1, 1), (0, 0)$ if $u_1 > 0, u_2 > 0$ and $(p_1^{NE}, q_1^{NE}) = (0, 1), (1, 0)$ if $u_1 < 0, u_2 < 0$. However, under our assumption, the game in Case 3.2 only has an NE in fully mixed strategies, and NEs in pure strategies do not exist. Therefore, we always have $u_1 u_2 < 0$ in Case 3.2 and the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$, which is also the NE point, is asymptotically stable.

For the other equilibrium points, based on conditions (3.65)–(3.68) and (3.69), the stability conditions become

$$r_{22} > r_{12}, c_{22} > c_{21} \quad \text{for } (0, 0) \tag{3.81}$$

$$r_{21} > r_{11}, c_{21} > c_{22} \quad \text{for } (0, 1) \tag{3.82}$$

$$r_{12} > r_{22}, c_{12} > c_{11} \quad \text{for } (1, 0) \tag{3.83}$$

$$r_{11} > r_{21}, c_{11} > c_{12} \quad \text{for } (1, 1) \tag{3.84}$$

As already noted, the game in Case 3.2 only has an NE in fully mixed strategies and we always have $u_1 u_2 < 0$. Then the inequalities (3.76) and (3.77) are true in Case 3.2. However, the stability conditions (3.81)–(3.84) for the equilibrium points $(0, 0), (0, 1), (1, 0), (1, 1)$ conflict with the inequalities (3.76) and (3.77). Therefore, the equilibrium points other than $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$ are unstable in this case.

Thus we can conclude that the NE point is asymptotically stable while the other equilibrium points are unstable in Case 3.2.

3.14.1 Simulation

We now simulate three matrix games to show the performance of the L_{R-I} lagging anchor algorithm. The first game is the matching pennies game. This game is a two-player zero-sum game and each player has two actions: heads or tails. If both players choose the same action, then player 1 gets a reward 1 and player 2 gets a reward -1. If the actions are different, then player 1 gets -1 and player 2 gets 1. Based on the reward matrix in Table 3.1(a) and the solutions in Example 3.1, the NE in this game is in fully mixed strategies such that each player plays heads and tails with a probability of 0.5. We set the step size $\eta = 0.001$ in (3.58) and $p_1(0) = q_1(0) = 0.2$. We run the simulation for 30,000 iterations. In Fig. 3-9, the players' probabilities of taking their first actions start from (0.2, 0.2) and move close to the NE point (0.5, 0.5) as the learning proceeds.

The second game we simulate is a two-player general-sum game, namely the prisoners' dilemma. In this game, we have two players and each player has two actions: defect or cooperate. A player receives a reward of 10 if it defects and the other player cooperates, or receives a reward of 0 if it cooperates and the other player defects. If both players cooperate, each player receives a reward of 5. If they both defect, each player receives a reward of 1. The reward matrix is shown in Table 3.3(b), where one player's reward matrix is the transpose of the other player's reward matrix. This game has a unique NE in pure strategies which is both players playing defect. We set the step size $\eta = 0.001$ in (3.58) and $p_1(0) = q_1(0) = 0.5$. We run the simulation for 30,000 iterations. Figure 3-10 shows that the players' strategies move close to the NE strategies (both players' second actions) as the learning proceeds.

Table 3.3 Examples of two-player matrix games.

(a) Matching Pennies	(b) Prisoners' Dilemma	(c) Rock-Paper-Scissors
$R_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$,	$R_1 = \begin{bmatrix} 5 & 0 \\ 10 & 1 \end{bmatrix}$,	$R_1 = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$,
$R_2 = -R_1$	$R_2 = (R_1)^{\mathrm{T}}$	$R_2 = -R_1$
NE in fully mixed strategies	NE in pure strategies	NE in fully mixed strategies

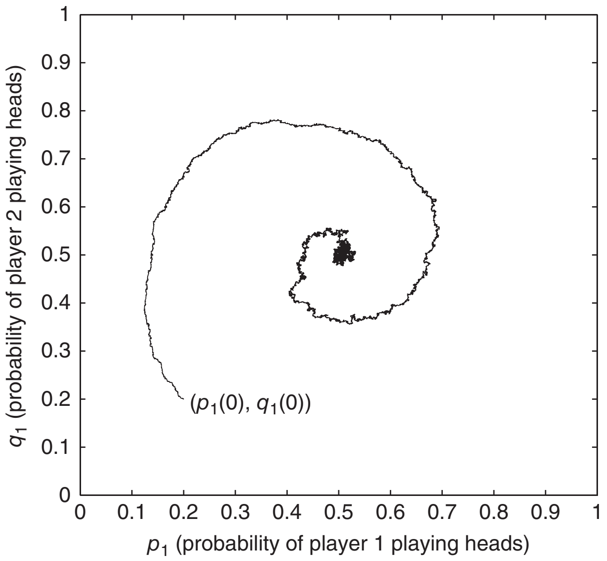

Fig. 3-9. Trajectories of players' strategies during learning in matching pennies. Reproduced from [8], © X. Lu.

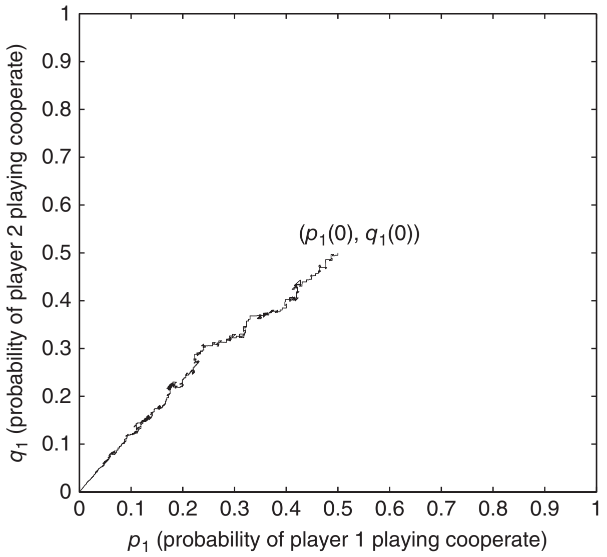

Fig. 3-10. Trajectories of players' strategies during learning in prisoners' dilemma. Reproduced from [8], © X. Lu.

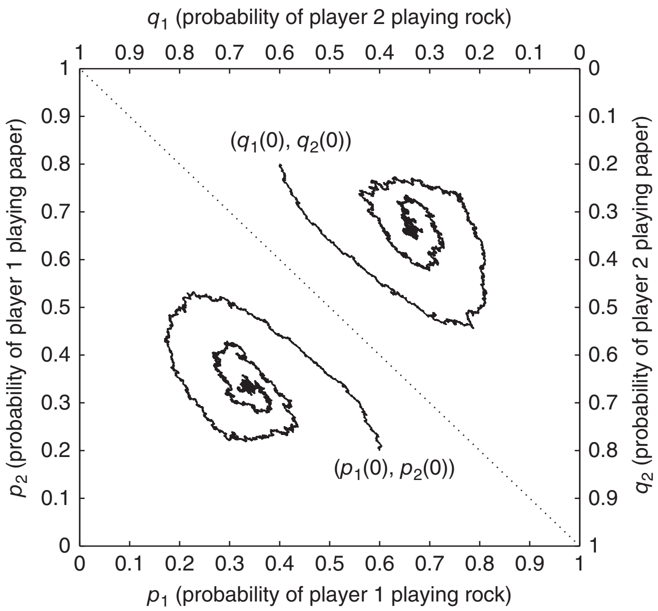

Fig. 3-11. Trajectories of players' strategies during learning in rock-paper-scissors. Reproduced from [8], © X. Lu.

The third game we simulate in this chapter is the rock-paper-scissors game. This game has two players and each player has three actions: rock, paper, and scissors. A winner in the game is determined by the following rules: paper defeats rock, scissors defeats paper, and rock defeats scissors. The winner receives a reward of 1 and the loser receives -1. If both players choose the same action, each player gets 0. The reward matrix is shown in Table 3.3(c). This game has an NE in fully mixed strategies which is each player choosing any action with the same probability of $1/3$. We set the step size $\eta = 0.001$, $p_1(0) = q_1(0) = 0.6$, and $p_2(0) = q_2(0) = 0.2$. We run the simulation for 50,000 iterations. Although we only prove the convergence for two-player two-action games, the result in Fig. 3-11 shows that the proposed L_{R-I} lagging anchor algorithm may be applicable to a two-player matrix game with more than two actions.

References

[1] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*. Cambridge, Massachusetts: The MIT Press, 1994.

[2] G. Owen, *Game Theory*. San Diego, California: Academic Press, 1995.

[3] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *Artificial Intelligence*, vol. 136, no. 2, pp. 215–250, 2002.

[4] T. Başar and G. J. Olsder, *Dynamic Noncooperative Game Theory*, *SIAM Series in Classics in Applied Mathematics*. London: Academic Press, 2nd ed., 1999.

[5] P. Sastry, V. Phansalkar, and M. Thathachar, "Decentralized learning of Nash equilibria in multi-person stochastic games with incomplete information," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 5, pp. 769–777, 1994.

[6] M. J. Osborne, *An Introduction to Game Theory*. Oxford: Oxford University Press, 2003.

[7] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in 11th International Conference on Machine Learning, (New Brunswick, United States), July 1994, pp. 157–163, 1994.

[8] X. Lu, Ph.D., "On Multi-Agent Reinforcement Learning in Games." Ph.D. Thesis, Carleton University, Ottawa, ON, 2012.

[9] S. P. Singh, M. J. Kearns, and Y. Mansour, "Nash convergence of gradient dynamics in general-sum games," in UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, California, USA, June 30 - July 3, 2000, pp. 541–548, 2000.

[10] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[11] E. Yang and D. Gu, "A survey on multiagent reinforcement learning towards multi-robot systems," in Proceedings of IEEE Symposium on Computational Intelligence and Games, 2005.

[12] L. Buşoniu, R. Babuška, and B. D. Schutter, "Multiagent reinforcement learning: a survey," 9th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 1–6, 2006.

[13] X. Lu and H. M. Schwartz, "An investigation of guarding a territory problem in a grid world," in American Control Conference, pp. 3204–3210, 2010.

[14] S. Lakshmivarahan and K. S. Narendra, "Learning algorithms for two-person zero-sum stochastic games with incomplete information," *Mathematics of Operations Research*, vol. 6, no. 3, pp. 379–386, 1981.

[15] S. Lakshmivarahan and K. S. Narendra, "Learning algorithms for two-person zero-sum stochastic games with incomplete information: a unified approach," *SIAM Journal on Control and Optimization*, vol. 20, no. 4, pp. 541–552, 1982.

[16] F. A. Dahl, "The lagging anchor algorithm: reinforcement learning in two-player zero-sum games with imperfect information," *Machine Learning*, vol. 49, pp. 5–37, 2002.

[17] F. A. Dahl, "The lagging anchor model for game learning—a solution to the crawford puzzle," *Journal of Economic Behavior & Organization*, vol. 57, pp. 287–303, 2005.

[18] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs, New Jersey: Prentice Hall, 1989.

[19] M. Thathachar and P. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Boston, Massachusetts: Kluwer Academic Publishers, 2004.