LEARNING SUCCESSFUL STRATEGIES IN REPEATED GENERAL-SUM GAMES

by

Jacob W. Crandall

A dissertation submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Brigham Young University

December 2005

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

Jacob W. Crandall

This dissertation 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 A. Ventura
Date	Kevin D. Seppi
Date	Bryan S. Morse
Date	Michael D. Jones

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of Jacob W. Crandall 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 K. Egbert
Graduate Coordinator

Accepted for the College

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

ABSTRACT

LEARNING SUCCESSFUL STRATEGIES IN REPEATED GENERAL-SUM GAMES

Jacob W. Crandall

Computer Science

Doctor of Philosophy

Many environments in which an agent can use reinforcement learning techniques to learn profitable strategies are affected by other learning agents. These situations can be modeled as general-sum games. When playing repeated general-sum games with other learning agents, the goal of a self-interested learning agent is to maximize its own payoffs over time. Traditional reinforcement learning algorithms learn myopic strategies in these games. As a result, they learn strategies that produce undesirable results in many games. In this dissertation, we develop and analyze algorithms that learn non-myopic strategies when playing many important infinitely repeated general-sum games. We show that, in many of these games, these algorithms outperform existing multiagent learning algorithms. We derive performance guarantees for these algorithms (for certain learning parameters) and show that these guarantees become stronger and apply to larger classes of games as more information is observed and used by the agents. We establish these results through empirical studies and mathematical proofs.

ACKNOWLEDGMENTS

I would be amiss without acknowledging the important roles that others have played in the development of this work. I certainly must acknowledge the significant role played by my advisor Michael A. Goodrich. Mike has provided me with invaluable insights and opportunities which have shaped this dissertation. My parents, siblings, and many teachers have also provided me with immense support throughout my life, to which I attribute so much of what I have been able to accomplish. Lastly, I acknowledge the vital support of my wife Tawna and daughter Gracie who have sacrificed much so that I could successfully complete this work.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

As technology advances, so does the need for artificial agents that can operate successfully in complex environments. However, obtaining successful behavior from artificial agents is a daunting task, since many of these environments are dynamic, complex, and may not be known in advance. Giving these agents the ability to learn is one way to overcome such challenges.

Endowing artificial agents with the ability to learn successfully in arbitrary situations is challenging. One reason for this difficulty is that many environments are non-stationary, such as those in which an agent interacts with other learning agents. Learning is particularly difficult in such environments since the consequences of an action may change over time. Additionally, an agent must determine how its actions affect the subsequent actions of other learning agents. Predicting these consequences is challenging because agents differ widely in the ways that they learn.

One common characteristic of real-life situations is that they repeat, sometimes many times, which results in repeated interactions between agents. These repeated interactions, whether between humans or machines (or both), should lead to improved behavior, since they give agents opportunities to learn. Thus, in many situations, multiagent learning inherently involves repeated play[1]. In these situations, agents should learn to act in ways that *improve their own payoffs over time*, which means that agents must be concerned not only with the immediate consequences of their actions, but also with their long-term effects.

The situations in which agents repeatedly interact vary widely. A common situation (or game[2]) is one in which a loss to one agent is the gain of another agent. These games are called zero-sum games. A second kind of situation arises when the joint behavior of the agents favors all agents equally. Such games are called fully cooperative games, and require only that agents learn to coordinate their actions appropriately. Between the two extremes of zero-sum games and fully cooperative games are a large class of games in which agents have conflicting interests. Frequently, such games

[1]The term *repeated play* is a term from game theory that suggests that the same game is played repeatedly by the same agents.

[2]We assume, as in game theory, that real-world situations can be encoded as games.

mean that learning agents must learn to cooperate with each other if they want to be successful. These three kinds of games constitute repeated general-sum games.

Regardless of the game structure, the goal of a (self-interested) agent in a repeated game is to maximize its own payoffs over time. Thus, three qualities determine the success of a learning agent: a) the quality of the strategies the agent eventually learns to play, b) the speed at which the agent learns them, and c) the classes of agents with whom the agent can effectively associate. While all three criteria are important, performing well in one criterion may come at the expense of one of the others. For example, an agent may choose to learn slowly in order to ensure that it learns correctly.

In addition to the three criteria just mentioned, a critical attribute of a multiagent learning algorithm is the knowledge that it requires of its environment. Due to sensor quality, among other things, an agent may not be able to know the actions taken by other agents, the learning methods used by other agents (if the agents are learning at all), the payoffs received by other agents, or even the payoffs that it receives itself[3]. Nevertheless, if an agent desires to learn successfully in these environments, it must be able to do so despite various knowledge deficiencies about its environment.

To date, multiagent learning algorithms have been largely unsuccessful at learning in general-sum games as a whole [52]. One reason for this failure is that most multiagent learning algorithms have been designed with the purpose of learning game theoretic solution concepts. While these solution concepts are important, they often make assumptions that do not apply to all infinitely repeated games between learning agents. As a result, these multiagent learning algorithms often learn myopic (and unprofitable) behavior in many theoretically important repeated games.

In this dissertation, we will discuss and develop multiagent learning algorithms that learn less myopic (and, thus, more profitable) solutions in infinitely repeated general-sum games.

1.2 Thesis Statement

When playing repeated general-sum games with other learning agents, the goal of a self-interested learning agent is to maximize its own payoffs over time. Algorithms can be developed that learn (when associating with other learning agents) to play strategies that produce significantly higher average payoffs over time than existing algorithms in many important infinitely repeated general-sum games. Performance guarantees for these algorithms can be established for certain learning parameters, and these guarantees become stronger and apply to larger classes of agents as more information is observed and used by the agents. These results can be established through empirical studies and mathematical proofs.

[3]We maintain that, if an agent is to learn, it must be able to determine the payoffs that its actions yield. However, the degrees to which an agent knows its payoffs may vary.

1.3 Contents

The outline of this dissertation is as follows. In Chapter 2, we give important background information and review related work. Specifically, we discuss game theoretic concepts and other related topics, review existing multiagent learning algorithms, and discuss general-sum games in more detail.

In Chapter 3, we consider the case in which an agent knows its own actions and can observe its own payoffs, but not those of its associates. We analyze an existing algorithm from the literature [70], and prove that it converges with high probability when associating with copies of itself[4] to near pareto efficient[5] solutions in all general-sum matrix games, provided that specific conditions are met. We show that these conditions can be satisfied in most infinitely repeated general-sum matrix games (and all 2-player matrix games). A preliminary version of this work was published in [24].

In Chapter 4, we present a trembling-hand version of the algorithm discussed in Chapter 3. We show (empirically) that this algorithm learns in self play to satisfy the conditions necessary for convergence (with high probability) to near pareto efficient solutions in general-sum matrix games if a) the conditions can be satisfied and if b) all agents benefit from doing so. Preliminary work for this chapter was published in [24].

We consider in Chapter 5 the case in which an agent can observe the actions of other agents. Under these conditions, a learning agent should possess two properties. First, an agent should never learn to play a strategy that produces average payoffs less than the minimax value of the game. Second an agent should learn to cooperate/compromise when doing so increases its payoffs. No learning algorithm from the literature is known to possess both of these properties without having knowledge of the payoffs of other agents. We present a reinforcement learning algorithm that provably satisfies the first property in all general-sum matrix games and empirically displays (in self play) the second property in a wide range of games. This chapter is an extended version of a paper published in the 22$^{\text{nd}}$ International Conference on Machine Learning [20].

The algorithm presented in Chapter 5 is designed only for matrix games. In Chapter 6, we extend this algorithm to the more general case of stochastic games. We show (empirically) that this algorithm continues to possess the same two properties (i.e., security and cooperate/compromise) in several important stochastic games. Multiagent learning algorithms in the literature do not satisfy these properties in many of these games.

In Chapter 7, we summarize the results from the previous chapters and discuss future work.

In the main body of the dissertation, we do not consider the case in which an agent

[4]We refer to situations in which an agent associates with copies of itself as *self play*.

[5]We will describe this term in more detail in Chapter 2. In short, a solution is pareto efficient if no other solution benefits one agent without hurting another agent.

knows the payoffs of other agents. However, when this information is available, an agent has the possibility of learning/associating more effectively with a larger class of agents. In Appendix A, we present a separate algorithm for such situations and show empirical results that suggest that this algorithm can learn pareto efficient solutions with a much larger class of agents. As this work is still considered preliminary in nature, we do not include it in the main body of this dissertation. This work was published in [22].

Chapter 2

Background and Related Work

In this chapter, we give necessary background information for the work of this dissertation. Additionally, we will review some of the more relevant work found in the literature.

Specifically, this chapter will proceed as follows. In the next section, we formally define the multiagent learning problem. Many of the central issues of the multiagent learning problem are related to game theory. Hence, we review important aspects of game theory in Section 2.2. However, game theory assumes that all agents are rational, which is certainly not true of all learning agents. Thus, in Section 2.3, we outline some alternate perspectives of decision making. In Section 2.4, we review past related work in multiagent learning. We review general-sum games in more detail in Section 2.5. In Section 2.6, we discuss the kinds of important information an agent may or may not know when it is learning. Finally, we provide a brief discussion about the current state of multiagent learning in the last section of this chapter.

2.1 The Multiagent Learning Problem

In this section, we formalize the multiagent learning problem. Let I be a set of n agents. Each agent i can choose from a set of actions denoted by A_i. Let $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, where $a_i \in A_i$, be a joint action for the n agents specified by I, and let $A = A_1 \times \cdots \times A_n$ be the set of possible joint actions of these agents. Let $a_i^t \in A_i$ denote the action played by agent i at time t, and let $\mathbf{a}^t = (a_1^t, a_2^t, \ldots, a_n^t)$ be the vector denoting the joint action played by the n agents at time t. Also, let $\mathbf{a}^t(J)$, where $J \subseteq I$, be the joint actions of the $|J|$ agents in J. Let $\mathbf{a_{-i}}$ be the joint actions of all agents except agent i.

Central to multiagent learning is the payoff matrix. Let $R = \{R_1, \cdots, R_n\}$ be the set of payoff matrices of the game, where R_i is the payoff matrix of agent i. Let $r_i(\mathbf{a})$ be the payoff to player[1] i when the joint action $\mathbf{a}$ is played, and let r_i^t be the payoff to player i at time t. Also, let $r_i(a_i, \mathbf{a}_{-i})$ be the payoff to player i when it plays a_i and the rest of the agents play $\mathbf{a}_{-i}$.[2]

[1] We use the terms player and agent interchangeably.

[2] More general definitions related to payoff matrices can be given. This definition assumes a deterministic payoff matrix. However, this definition will suffice for motivating the work of this dissertation. We will discuss a more general definition in Chapter 5.

	a	b
A	1, 1	-1, 2
B	2, -1	0, 0

Table 2.1: An example of a 2-agent, 2-action matrix game.

We are now ready to formally define what is meant by the term *matrix game*.

Definition 2.1. *(Matrix Game) A matrix game is the tuple* (I, A, R)*, where* I*,* A*, and* R *are defined previously.*

As we will be referring to matrix games extensively throughout this dissertation, we will display them in the format seen in Table 2.1. In this matrix game $I = (\text{player } 1, \text{player } 2)$. Player 1 is referred to as the row player and player 2 is the column player (since player 1 selects a row and player 2 selects the column). Thus, $A_1 = \{A, B\}$, $A_2 = \{a, b\}$, and $A = \{(A, a), (A, b), (B, a), (B, b)\}$. Each cell in the game matrix (shown in Table 2.1) shows the payoffs that are rewarded to the agents for the unique joint action specified by the row and column. The payoff to the row player (player 1) is listed first, followed by the payoff to the column player (player 2). For example, if the row player plays B and the column player plays a (which is the joint action (B, a)), then the row player receives a payoff of 2, while the column player receives a payoff of -1. Thus, the individual payoff matrices for this game (R_1 and R_2) are $R_1 = \begin{bmatrix} 1 & -1 \\ 2 & 0 \end{bmatrix}$ and $R_2 = \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix}$.

A *strategy* for agent i is a distribution π_i over its action set A_i. Such a strategy may be a *pure strategy* (where all probability is placed on a single action) or a *mixed strategy* (otherwise). The joint strategy played by the n agents is $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$ and, thus, $r_i(\boldsymbol{\pi})$ is the expected payoff for agent i when the joint strategy $\boldsymbol{\pi}$ is played. A *solution* (or *outcome*) is a particular joint strategy[3].

The task of a learning agent is to learn to play a strategy π_i such that its average payoff over time (denoted $\bar{r}_i$ for agent i) is maximized. Let $\bar{r}_i$ be given by

$$\bar{r}_i = \frac{1}{T} \sum_{t=1}^{T} r_i(\pi_i^t, \boldsymbol{\pi}_{-i}^t) \tag{2.1}$$

where π_i^t is the strategy played by agent i at time t, $\boldsymbol{\pi}_{-i}^t$ is the joint strategy played by all the agents except agent i, and $1 \leq T \leq \infty$ is the number of *episodes* in the game.

We consider the case in which each round of the game is played between the same agents. Thus, the length of the game (i.e., T) is important since it affects how an

[3]While a solution is a joint strategy, we often refer to a solution as the joint payoff that it produces.

agent should behave. For example, in the one-shot game (i.e., when $T = 1$) the agents need not consider how their current action will affect the subsequent actions of the other agents. Thus, they should play so as to maximize the reward that they will receive in that round. On the other hand, if $T \to \infty$ (known as an infinitely repeated game), the agents should consider how their actions will affect the actions of the agents in subsequent rounds. When T is somewhere in between these two extremes, the agents must find a balance between maximizing rewards in the current episode and considering how actions will affect subsequent rewards. We note that the agents may not know T. In these situations, the agents may believe that play will terminate with probability γ after each episode.

While successful agents should have the ability to learn to act successfully for all values of T, we focus in this dissertation on infinitely repeated games or games in which the probability that play will continue after each episode (i.e., γ) is close to unity.

In this dissertation, we only consider simultaneous move games. In each episode[4] of a simultaneous move matrix game, each agent selects an action without knowledge of the actions selected by the other agents. The agents are then rewarded based on the resulting joint action. We note that a second kind of game is a turn-taking game in which the agents take turns selecting actions. While some argue that turn-taking games are more representative of real world situations [10], we note that stochastic games, which we will discuss in Chapter 6, can be designed so that agents must consider both simultaneous and sequential (turn-taking) decision making.[5]

2.2 Game Theory

Game theory seeks to explain how agents (should) act so as to maximize their payoffs when they are associating with other agents. As such, it is relevant to the multiagent learning problem (at least to the degree that it does explain how agents should act). In this section, we give a brief overview of the concepts and results that have had the most influence on the multiagent learning problem to date. A more in-depth treatment of game theory can be found, among other places, in [35].

2.2.1 Game Theory's Big Assumption: Rationality

In game theory, all agents are assumed to be *rational*. That is, game theory assumes that each agent will play so as to maximize its own payoffs over time. Implicit with this assumption is the assumption that agents have unlimited resources for

[4] An episode of a matrix game is simply the play of a single joint action and the assignment of the subsequent payoffs

[5] Stochastic games can represent (if designed as such) a combination of simultaneous and turn-taking games only when the actions of other agents can be observed after each joint action taken by the agents.

determining a course of action. This assumption is frequently violated. As a result, an agent's associates often behave (seemingly) irrationally. Thus, game theoretic concepts may dictate behavior that results in low average payoffs over time.

A second drawback of game theory (and its assumption of rationality) is that it cannot always define what is *best*. This is because many situations have multiple equilibria[6]. In such situations, the agents may not agree on which equilibrium rational agents should learn to play. Thus, rationality does not always define how an agent should behave.

Despite these drawbacks, game theory has played an important role in many multiagent learning algorithms. Specifically, the behavior of these algorithms can sometimes be described by game theory. Additionally, game theory has defined important goals for multiagent learning algorithms to attain.

2.2.2 Equilibrium Concepts

Central to game theory are equilibria. That is, the play of rational agents should converge to some strategy. Perhaps the most important of these equilibrium concepts is the Nash equilibrium.

The Nash Equilibrium

The *Nash equilibrium* (NE) has had the most impact on the design and evaluation of multiagent learning algorithms to date. The concept of a Nash equilibrium is closely tied to the *best response*. Formally, the strategy π_i^* is a best response for agent i if $r_i(\pi_i^*, \boldsymbol{\pi}_{-i}) \geq r_i(\pi_i, \boldsymbol{\pi}_{-i})$ for all possible π_i. When all agents play best responses to the strategies of other agents, the result is an NE. Thus, in an NE, no agent has an incentive to unilaterally change its strategy. Nash showed that every game has at least one NE [60]. However, there is no known algorithm for calculating NEs in polynomial time [61].

While a self-interested agent should play a best response to the strategies of other agents (and, thus, all agents should play a best response resulting in an NE), many games have multiple NEs. In fact, many games have an infinite number of *NEs of the repeated game* (rNEs), which we will discuss shortly. Thus, the task of a learning agent does not end with learning a best response. Rather, a learning agent must influence its associates to play so that its best response is as profitable as possible.

One-Shot Nash Equilibrium

When only the payoffs of the current episode (in matrix games) are considered, an NE is said to be a *one-shot NE*. While the one-shot NE is meant in general for one-shot games, it has frequently been a goal of multiagent learning algorithms in

[6]We discuss equilibrium concepts shortly.

repeated games (e.g., [9, 41, 52, 18, 8]). This is not completely without justification, as all one-shot NE are NEs of the repeated game[7].

Minimax

Like the one-shot NE, the *minimax solution*, another central result of game theory, is calculated over the payoffs of the game matrix. The minimax solution (or strategy) for agent i (denoted π_i^m), which is the strategy that maximizes the minimum payoff that an agent can receive, is given by

$$\pi_i^m = \operatorname*{argmax}_{\pi_i} \min_{\boldsymbol{a}_{-i}} r_i(\pi_i, \boldsymbol{a}_{-i}) \tag{2.2}$$

Note that the minimax solution can be a pure or mixed strategy. However, the min operator need only be over pure strategy solutions (i.e., the joint actions in the set A_{-i}). The minimax value for agent i (denoted m_i) is the average payoff received by agent i when it plays its minimax strategy and the other agents seek to minimize its payoffs. Formally, the *minimax value* for agent i is given by

$$m_i = \max_{\pi_i} \min_{\boldsymbol{a}_{-i}} r_i(\pi_i, \boldsymbol{a}_{-i}) \tag{2.3}$$

The minimax solution is intriguing for several reasons. First, in constant-sum games (games in which the payoffs of the agents for any given strategy sum to a constant), the minimax solution is the unique NE of the game (regardless of whether the game is repeated or one-shot). However, the minimax solution is not a one-shot NE in all general-sum games, although it is an NE of the repeated game. Nevertheless, the minimax value is the highest payoff an agent can expect to receive without the cooperation of its associates, regardless of the game.

Unlike the NE, the minimax solution and value can be computed via linear programming [51].

Pareto Efficiency

Before discussing rNE, we must first discuss the concept of pareto efficiency. To do so, we first define what it means to be *pareto dominated*, a term we will use to define pareto efficiency.

Definition 2.2. *(Pareto Dominated) A solution $\boldsymbol{\pi}$ is strictly pareto dominated if there exists a joint action $\mathbf{a} \in A$ for which $r_i(\mathbf{a}) > r_i(\boldsymbol{\pi})$ for all i and weakly pareto dominated if there exists a joint action $\mathbf{a} \in A$ for which $r_i(\mathbf{a}) \geq r_i(\boldsymbol{\pi})$ for all i.*

Definition 2.3. *(Pareto Efficient) A solution $\boldsymbol{\pi}$ is weakly pareto efficient (PE) if it is not strictly pareto dominated and strictly PE if it is not weakly pareto dominated.*

[7] This result can be obtained from the folk theorem, which we discuss shortly.

	cooperate	defect
cooperate	1, 1	-1, 2
defect	2, -1	0, 0

Table 2.2: Payoff matrix for prisoner's dilemma game.

We note that these definitions specify that the solution $\boldsymbol{\pi}$ need only be pareto efficient with respect to pure strategies in order to be considered pareto efficient. This means that the joint payoff corresponding to a pareto efficient solution $\boldsymbol{\pi}$ may very well reside in the interior of the game's convex hull.

Another point about pareto efficiency that deserves mentioning is that pareto efficiency should not, ultimately, be the only goal of a self interested agent. This is because many pareto dominated solutions can give a single agent higher payoffs than some pareto efficient solutions. However, in games between successful learners, play should converge to pareto efficient solutions. This is particularly true of self play. Otherwise, the average expected payoff of an agent will be lower than it need be. We note several learning algorithms have included pareto efficiency as a goal, including those described in [67, 62]).

The Folk Theorem and Nash Equilibrium of the Repeated Game

The *folk theorem* for repeated games is an important part of game theory (although it has been largely ignored in portions of the multiagent learning community until recently). A good description of it can be found in, among other places, [35]. Here, we review it and discuss its implications to multiagent learning in repeated general-sum games.

Consider the prisoner's dilemma matrix game shown in Table 2.2. The joint payoff space of the prisoner's dilemma game is shown in Figure 2.1. In the figure, the x-axis shows the payoffs to the row player and the y-axis shows the payoffs to the column player. Convex combinations of the various joint payoffs of the game form the game's payoff space (defined by the game's convex hull), which is the union of the shaded regions (light and dark) in the figure. Since each player can guarantee itself a non-negative payoff by playing *defect*, neither of the players has incentive to receive an average payoff (over time) less than 0. Thus, the darkly shaded region in the figure shows the set of joint payoffs that the agents may possibly accept as reasonable average payoffs per episode of the game. The folk theorem says that any joint payoff in this region can be sustained by an rNE, provided that the discount rates of the players are close to unity (i.e., players believe that play will continue with high probability after each episode).

More formally, let C be the set of points contained in the convex hull of the game. Any joint payoff $(r_1, r_2) \in C$ such that $r_1 \geq m_1$ and $r_2 \geq m_2$ can be sustained by an

Figure 2.1: Payoff space of the prisoner's dilemma game shown in Table 2.2.

rNE provided, again, that the discount rates of the players are close to unity. Note that this means that all one-shot NEs are rNEs, as all one-shot NEs do not give an agent a payoff below its minimax value. The folk theorem can be extended to games of more than two players [35].

The folk theorem shows that there are an infinite number of rNEs in all games in which the joint payoff $m = (m_1, m_2)$ is not pareto efficient (which constitutes a large portion of general-sum games). Thus, while the folk theorem identifies the existence of rNEs, it does little to identify which equilibrium (and strategies) that agents should learn to play. It does, however, help to identify the types of payoffs an agent can expect to receive. Since a self-interested agent should seek to maximize its own payoffs over time, it should not, in most cases, be content to play its part of an NE in which it only receives its security level (m_i). Whether this is the case largely depends on its associate(s), but in many games, such as the prisoner's dilemma, all agents may achieve payoffs higher than their security levels if they are willing to cooperate. In most circumstances, we believe that, if all agents learn successfully, they should learn to play a pareto efficient rNE.

Littman and Stone [53] give an algorithm to compute, in polynomial time, a particular rNE for two agent games. In their algorithm, they select an rNE that yields payoffs that maximize the product of the agents' advantages[8]. This type of equilibrium solution is suggested by Nash [59].[9]

While maximizing the product of the agents' advantages is a reasonable target rNE (it is pareto efficient and produces reasonably high payoffs for each agent), it may not be the solution that a self-interested agent would want to adopt. To see this, consider the matrix game chicken shown in Table 2.3. The game's payoff space

[8]Littman defines agent i's advantage for the joint action $\mathbf{a}$ as $r_i(\mathbf{a}) - m_i$.

[9]The Nash bargaining solution depends on four axioms, one of which is the independence of irrelevant alternatives. Littman's algorithm appears to violate this axiom.

	swerve	straight
swerve	3, 3	2, 3.5
straight	3.5, 2	1, 1

Table 2.3: Payoff matrix for the game chicken.

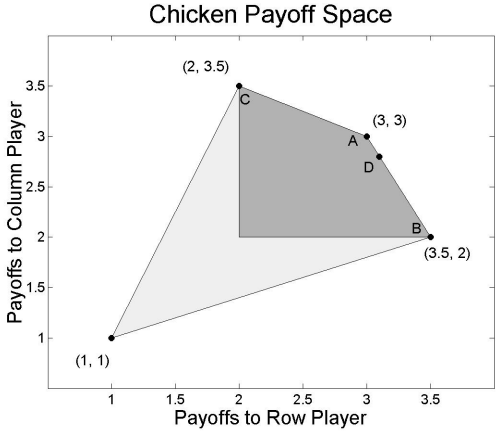


Figure 2.2: Payoff space of the game chicken shown in Table 2.3.

is shown in Figure 2.2. In chicken, each agent can guarantee itself at least 2 points per episode if it always plays *swerve*. The one-shot NEs of chicken are for the row player to play *swerve* and the column player to play *straight* and vice versa (labeled as points C and B in Figure 2.2, respectively). However, when the game repeats many times, a self-interested agent might be a fool to always accept a payoff of 2, since there are many rNE that give it higher payoffs. Littman's algorithm calculates that both agents should always *swerve* resulting in a payoff of 3 for each agent (labeled A in Figure 2.2). However, if the column player will continue to swerve as long as the row player plays *straight* no more than every fifth play, then the row player should learn to play *straight* every fifth play. This would result in the more desirable (for the row player) average joint payoff of $(3.1, 2.8)$ (shown as point D in Figure 2.2). However, learning to play *straight* every fifth play could result in lower payoffs to the row player over time if the column player will not "accept" this behavior. Therefore, the solution that the row player should learn to play is unknown without an understanding of how the column player will behave.

Correlated Equilibrium

Another powerful equilibrium concept is that of *correlated equilibrium* (CE) [1]. To understand CEs, consider the case in which each agent observes a signal. The signals observed by the agents may be correlated or independent. If each agent plays a best response to the strategies of the other agents *given its observed signal*, then

the resulting strategy is a CE. Thus, in a CE, the signal dictates which strategies the agents should play. A simple example of a CE is a traffic signal. When two agents meet at an intersection (going in orthogonal directions), the agents receive the signal (from the traffic light) of either `<GO,STOP>` or `<STOP,GO>`. It is a CE for each agent to obey its signal, as it is a best response (given that the other agent obeys its signal) for each agent to do so. A formal definition of CEs is given in [1] (among other places).

When the signals observed by the agents are not correlated, CEs are NEs. Thus, CEs are a superset of NEs. However, unlike NEs, CEs can be computed via linear programming. Additionally, while no algorithm is known to converge to NEs in general, no-regret algorithms (which we will discuss shortly) [31] do converge to CEs [40].

2.3 Alternate Perspectives

While game theory is a powerful method for describing the behavior of rational agents, not all agents are rational (at least from the perspective of game theory). However, many seemingly *irrational* agents are successful; more so, sometimes, than rational agents (which calls into question the rationality of so called rational agents).

One such agent is copy-cat algorithm tit-for-tat (TFT) [2], which was designed for the iterated prisoner's dilemma. In this game, TFT simply plays the action its associate played on the previous episode. This strategy is, from the perspective of game theory, irrational, since it would cooperate even when it would be beneficial for it to defect. However, Axelrod found that TFT performed very well in iterated prisoner's dilemma tournaments when associating with a large class of agents. In fact, in the tournaments conducted by Axelrod, TFT came out the winner.

Because of repeated examples (such as that of TFT) of seemingly irrationality among successful agents, alternate theories of decision making have been developed. Some of these theories seek to do completely away with the rationality assumption. Others seek to modify it slightly or to add to it. In the rest of this section, we present several of these alternate theories for decision making.

One alternate theory of decision making is known as *bounded rationality*, a term coined by Herbert Simon [68]. Bounded rationality says that an agent's rationality is bounded by time and knowledge constraints. Thus, rather than seeking to maximize payoffs, agents seek instead to obtain payoffs that are good enough, or *satisficing*.

Stirling furthered the notion of satisficing by defining a *satisficing equilibrium* [74]. A satisficing equilibrium is a joint strategy that yields a result that is *good enough* for all agents (i.e., a joint strategy that is satisficing to all agents). While Simon uses an aspiration-based model, Stirling and his associates argue for a satisficing model with dichotomous attributes [74, 73, 58, 38].

Satisficing with dichotomous attributes is related to the idea of suspending judgment presented by Levi [48, 49]. Levi argues that when an agent is presented with

two conflicting criteria, it must withhold making a judgment (i.e., keep the two criteria separate) until sufficient information can be gathered to resolve the conflict. This methodology flies in the face of traditional approaches, which combine the two criteria into one before making a decision.

Many have argued that bounds on rationality mean that agents must use heuristics (based on what they already know) to make decisions [34, 64]. Related to this argument is Damasio's somatic marker hypothesis which discusses the role of emotions in the decision making process [25, 26]. Other theories for decision making include *psychological game theory* [17], *behavioral game theory* [11], and Frank's commitment model [32]. Others argue that altruistic behavior, not self-interest, largely governs agents within societies. This, they argue, helps explain why agents cooperate when they do not necessarily benefit from doing so. A large collection of articles on this subject can be found in [30].

2.4 Existing Learning Algorithms

In this section, we overview related work in multiagent learning. Because of the vast number of multiagent learning algorithms found in the literature, we must restrict our attention to those that have had the most impact on the multiagent learning community as well as to those that seem to be particularly connected to the work presented in this dissertation. We divide the learning algorithms that we review into four different (although related) categories: belief-based learning, reinforcement learning, no-regret learning, and principle-based learning.

2.4.1 Belief-based Learning

Belief-based agents construct models of the strategies employed by their associates. They then use these beliefs to calculate an optimal policy. Many such algorithms exist, the most studied of which are Fictitious Play [33] and Bayesian learning (see, for example, [45]).

Fictitious Play

A fictitious play agent models the strategies of its associates by recording the empirical distribution of their actions. The agent then plays a best response to its beliefs about the other agents' strategies. It provably converges (in self play) to a NE in games that are iterated dominance solvable[10], and does fairly well in learning one-shot NEs in other games as well (see, for example, [43]). However, it is known to not converge in games such as Shapley's game (which is shown in Table 2.4). A number of other drawbacks of fictitious play include a) it assumes that transitions

[10]Iterated dominance solvable games are games that can be solved by iteratively removing actions that are dominated.

	a	b	c
A	$0, 0$	$0, 1$	$1, 0$
B	$1, 0$	$0, 0$	$0, 1$
C	$0, 1$	$1, 0$	$0, 0$

Table 2.4: Shapley's Game.

and reward functions are known for all players (i.e., huge knowledge requirements), and b) it, like many other multiagent learning algorithms, generally performs poorly in repeated general-sum games in which no one-shot NE is pareto efficient.

An interesting result regarding belief-based agents such as fictitious play is given in [71]. In this work, Stimpson *et al.* show that learning models such as fictitious play are unable to adapt their behavior to the strategies of associates in a specific public goods game [11]. Thus, the behavior of such agents in this game is essentially reduced to a purely random strategy.

Bayesian Learning

Fictitious play is a special case of Bayesian learning [33]. General cases of Bayesian learning have been studied in multiagent systems, perhaps most notably in [45]. In this work, Kalai and Lehrer show that Bayesian learners that know their own payoff matrices must eventually play according to an rNE, as long as the individuals' initial beliefs about the play of their associates are compatible with the true strategies chosen their associates. However, no guarantee is placed on which rNE they will learn to play.

Another important (earlier) work on Bayesian learning is given by Aumann [1]. In this work, Aumann shows that Bayesian learning leads to convergence to correlated equilibria.

2.4.2 Reinforcement Learning

Reinforcement learning [44, 75] has frequently been used in multiagent learning algorithms. However, the use of reinforcement learning algorithms in multiagent environments is not justified mathematically since multiple learners make the environment non-stationary. These environments are non-stationary because the strategies of learning agents change over time. Nevertheless, reinforcement learning has frequently been used in multiagent contexts, and not without success (e.g. [16, 66]).

Q-learning

The most commonly used (and extended) reinforcement learning algorithm in multiagent learning has been Q-learning [76]. Since many of the algorithms we discuss

in this dissertation are based (to some degree) on this algorithm we now discuss it briefly.

In Q-learning, an agent seeks to learn the true expected discounted reward $Q^*(s, a)$ that it will receive for taking an action a from a particular state s. It does so by maintaining a Q-estimate $Q_t(s, a)$ for each state action pair (s, a). Each time that action a is taken from state s, the Q-estimate $Q_t(s, a)$ is updated using the following equation:

$$Q_t(s, a) = (1 - \alpha)Q_{t-1}(s, a) + \alpha[r + \gamma \max_b Q_{t-1}(s', b)] \tag{2.4}$$

where r is the immediate reward received, α is the learning rate, s' is the agent's subsequent state, and γ is the discount factor. It has been shown that, as long as each state-action pair (s, a) is visited/taken infinitely often and the learning rate is decreased slowly but not too slowly, $Q_t(s, a)$ will converge in stationary environments to $Q^*(s, a)$ [76]. However, since environments affected by multiple learners are non-stationary, these convergence guarantees do no apply to Q-learners associating with other learners in repeated games.

While Q-learners typically learn slowly, they require little knowledge of their environment. A Q-learner requires only that it know the actions it can take from each state and the immediate rewards that it receives when it takes those actions. It does not require a model of its environment (i.e., state transition probabilities and reward structure).

Like fictitious play, Q-learning agents and their variants do not typically learn desirable policies when they play (with other learners) repeated games in which no one-shot NE is pareto efficient. One exception to this is found in a Q-learning algorithm designed by Sandholm and Crites [66] for use in an iterated prisoner's dilemma. Sandholm and Crites showed that the Q-learning algorithm can be made more effective when the actions of other agents are observable, since these actions can be incorporated into the agent's state (see also [16, 57]). In such cases, they found that Q-learners are sometimes able to learn mutual cooperation in the iterated prisoner's dilemma in self play (although their results show that defection is still frequently learned by these learners).

Because Q-learning is not suited for multiagent environments, a number of extensions of the Q-learning algorithm to multiagent domains have been attempted. We now review these algorithms[11].

Minimax-Q

Since Q-learning is not mathematically designed for multiagent contexts, it has been adapted in several interesting ways. The first of these adaptations, called

[11] We note that these algorithms are joint-action learners [16]. Joint-action learners learn based on the joint strategies of the agents rather than their individual strategies. Joint-action learners must be able to observe the actions of other agents.

minimax-Q, was presented by Littman [51] for two-agent zero-sum games. Littman replaced the max operator in Equation (2.4) (i.e., $\max_b Q_{t-1}(s', b)$) with a minimax operator over the Q-values of the subsequent state s'. He proved [55] that this technique converges to the game theoretic optimal value (minimax solution) for two-agent zero-sum games. It is not, however, effective in exploiting inferior agents.[12] Minimax-Q also requires knowledge of the actions of other agents.

Nash-Q

A second variation of Q-learning, known as *Nash-Q* in the literature, was proposed by Hu and Wellman [41]. Hu and Wellman targeted multiagent learning in general sum games by replacing the max operator in Equation (2.4) with the payoff associated with a NE of the Q-values of the subsequent state s'. In order to do so, they required that the payoffs and actions of all agents in the game be known (so that the Q-values of all agents in the game can be calculated). The algorithm provably converges to a one-shot NE solution (again, over the Q-values) in a limited set of games [41, 7].

Friend-or-Foe Q-learning

A third variation of multiagent Q-learning, called friend-or-foe Q-learning, was proposed by Littman [52]. Friend-or-foe Q-learning targets adversarial and fully cooperative games. In this algorithm, an agent labels each associate as a friend or a foe. If an agent is a foe, then Q-values are updated as in the minimax-Q. If an agent is labeled a friend, then the max operator in Equation (2.4) is replaced with $\max_{a_1 \in A_1,\ a_2 \in A_2} Q(s', a_1, a_2)$ (for two agent games), where A_1 and A_2 are the set of actions in state s' available to agents 1 (self) and 2 (friend) respectively. If associates are labeled correctly, the algorithm is guaranteed (in self play) to converge to a one-shot NE in both adversarial and fully cooperative games. In fully cooperative games, friend-Q learns (again, in self play) to play the one-shot NE that yields the highest payoff to all agents. Friend-or-foe Q-learning requires the same knowledge of the environment as minimax-Q.

Correlated Q-learning

A fourth variation on the Q-learning algorithm generalizes friend-or-foe Q-learning. This variation, by Greenwald and Hall [39], replaces the max operator in the Q-update equation with a mechanism that calculates a correlated equilibrium. While somewhat promising in some contexts, correlated Q-learning can lead to convergence to unprofitable equilibria. Additionally, calculating an appropriate correlated equilibrium is awkward and requires large knowledge requirements. These knowledge requirements include knowledge of the actions and payoffs of all agents.

[12] Traditional Q-learners are sometimes successful at exploiting inferior agents.

Wolf-PHC

An important work by Bowling and Veloso [9] further shows the difficulties associated with multiagent learning in general-sum games. Bowling and Veloso proposed two desiderata that a multiagent learning algorithm should be able to achieve: a) rationality, which they define as learning to play a best response to the stationary strategies of associates/opponents and b) convergence to a stationary policy. None of the previous algorithms presented in the literature fulfilled these two qualities in general-sum games. They proposed an algorithm, called WoLF-PHC (Win or Learn Fast Policy Hill-Climbing), that fulfills these properties in some important games (in self play). An agent need only know the actions it can take and the rewards that it receives in order to implement WoLF-PHC.

This work by Bowling and Veloso has given rise to a large variety of new algorithms, including those that seek to fulfill the rationality and convergence properties (of which [18] is a notable example). Additionally, this work has led to discussions of exploiter agents [14, 5], as well as a proposed improvement to the WoLF-PHC algorithm [4].

Learning Aspirations via Reinforcement Learning

While most reinforcement learning algorithms learn the payoffs associated with specific actions, a separate vein of reinforcement learning has focused instead on learning aspiration levels and then using the principle of satisficing to generate strategies [46, 72, 70, 71, 24]. This approach has lead to algorithms that learn near pareto efficient and fair solutions. We discuss these approaches in more detail in Chapters 3 and 4 of this dissertation.

2.4.3 No-regret Learning

No-regret algorithms (also known as universally consistent algorithms [33]) are becoming increasingly popular in the literature. Loosely, *regret* compares an algorithm's performance with that of its best strategy (or expert [50]). An algorithm is said to have no-regret if it performs at least as well (in the limit) as its best strategy. A formal description can be found in [31].

No-regret learning converges to NEs in dominance-solvable, constant-sum, and 2×2 general-sum games, but does not converge in Shapley's Game [43]. This result is reminiscent of fictitious play. However, no-regret learners require much less knowledge of the environment than does fictitious play.

A challenge of no-regret algorithms is determining the performance of unplayed strategies in order to measure regret. This is done by comparing a strategy against the empirical distribution of associates' actions. However, this approach often leads

	c	d
C	$3, 3$	$0, 5$
D	$5, 0$	$1, 1$

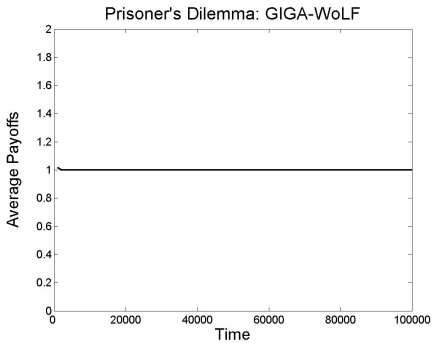


Figure 2.3: Average payoffs over time for GIGA-WoLF [8] (in self play) in the iterated prisoner's dilemma game shown at left. The payoff matrix played by the agents is shown at left.

to myopic behavior since it assumes that an agent's actions do not affect the future strategies of its associates. This assumption is not true of play between learning agents. Figure 2.3 demonstrates this myopic behavior in the iterated prisoner's dilemma for the no-regret learning algorithm called GIGA-WoLF [8]. The figure shows that GIGA-WoLF (quickly) learns to defect in self play, which yields a payoff of 1 to both agents. We note that this myopic (and pareto dominated) behavior is typical of most of the algorithms we have discussed in this chapter, and is not limited to no-regret algorithms.

Algorithms have been developed with intentions to overcome this problem while maintaining properties of no-regret [27, 15]. However, these algorithms have been primarily validated for games played with static agents (such as TFT) rather than games played with other learners.

2.4.4 Principle-based Learning

Of late, a number of algorithms have emerged that select strategies based on various principles other than on the best response. We make particular mention of algorithms using the (interconnected) principles of reputation, signaling, and teaching. Below, we outline some justifications for the use of these principles in multiagent learning as well as some algorithms that follow them.

Reputation

When multiple learning agents interact in repeated general-sum games, reputations [47, 56] may be established to improve payoffs (or decrease them, if undesirable reputations are established). However, agents must be careful that the *signals* used to build reputations can be perceived by the other learning agents, since attempting

to construct such reputations may be costly if they are not perceived [32]. We have done some preliminary work on the role of reputations in multiagent learning [23, 21].

Signaling

Agents send signals (whether explicitly or implicitly, intentionally or unintentionally) that establish their reputations. In his book *The Stag Hunt* [69], Skyrms identifies signaling as one of the three important issues for play to converge in social situations to mutually beneficial equilibrium points (rather than undesirable ones). Skyrms discusses how signals that contain sufficient information lead to the selection of certain equilibrium solutions over others. He calls such systems *signaling systems*.

Work on determining the importance of signals has been plentiful. Of particular relevance is [65], which analyzes the effect of signaling on pigeons playing the iterated prisoner's dilemma. However, the application of signals in multiagent learning algorithms has not been abundant, although Sen *et al.* [67] does address the matter. We have done some preliminary work involving implicit signaling in the iterated prisoner's dilemma [36, 23, 21]. We note also that work on correlated equilibria is tied closely to signaling.

Teaching

The principles of signaling and reputation both tie in closely to the principle of teaching. When a learning agent acts in repeated environments with other learners, it also acts as a teacher to the other agents in the system. However, most multiagent learning algorithms developed to date have not taken this into account, probably because of the complexity it appears to add to the learning process. However, teaching techniques have been used with success by non-learning (static) algorithms.

Littman and Stone [54] demonstrated how a couple of different kinds of "leader" algorithms can teach best-response learners to play various strategies in two-agent, two-action canonical games. The first, called *Bully*, calculates a strategy that induces best-response agents to play the Nash equilibrium of the stage game most suited to the *Bully* agent. The second agent, called *Godfather*, computes a revenge strategy (tit-for-tat in the prisoner's dilemma) to be used if the other agent does not play cooperatively. Both leader algorithms demonstrate superior performance in selected games against certain associates, although they both can perform poorly in self-play. In concluding their paper, Littman and Stone conclude that combinations of leader and best-response qualities should be incorporated into learning agents. This work is the foundation for several learning algorithms in the literature [63, 62, 22].

Other examples of teaching in the literature include Camerer *et al.*'s strategic teacher [12], work by Baker and Rachlin [3], and TFT. In all these cases, the "teaching" is done by a static agent rather than a learner. We have presented other preliminary results on how teaching methods can be incorporated in learners in [23, 21, 22].

Matching Pennies

	a	**b**
A	1, -1	-1, 1
B	-1, 1	1, -1

Rock, Paper, Scissors

	r	**p**	**s**
R	0, 0	-1, 1	1, -1
P	1, -1	0, 0	-1, 1
S	-1, 1	1, -1	0, 0

Table 2.5: Two simple competitive matrix games, called zero-sum games. The loss to one agent is the gain to the other.

We note that a serious drawback of algorithms using these principles is that they, like game theory, often assume knowledge of the actions and payoffs of other agents.

2.5 General-Sum Games

In this dissertation, we will analyze multiagent learning algorithms in repeated general-sum games. As we mentioned in Chapter 1, general-sum games can be divided into three classes or categories: competitive games, cooperative games, and games of conflicting interest. In this section, we briefly discuss the main characteristics of each category. We also provide example matrix games for each category that have generated particular interest in the literature. While we present only matrix games in this chapter, we note that versions of many of these games also exist in stochastic (multi-state) games, which we discuss in Chapter 6.

2.5.1 Competitive Games

In competitive games, a loss to one agent is the gain of another agent. Competitive games have no solution in which all agents receive an average payoff greater than their minimax value m_i. Two simple (yet popular) examples of competitive matrix games are shown in Table 2.5. Both of these games have a single rNE (a one-shot NE) which is the minimax solution. Thus, in these games, an agent should receive an average payoff of m_i (provided, of course, that its opponent is unexploitable).

2.5.2 Cooperative Games

Cooperative games are exactly opposite of competitive games. In these games, all agents share common goals, some of which may be more profitable than others. Table 2.6 shows a fully cooperative game with two one-shot NEs ((A, a) and (B, b)), and an infinite number of rNEs. The NE (A, a) is the most desirable outcome.

	a	b
A	4, 4	0, 0
B	0, 0	2, 2

Table 2.6: A fully cooperative matrix game.

Prisoner's Dilemma

	c	d
C	3, 3	1, 4
D	4, 1	2, 2

Chicken

	c	d
C	3, 3	2, 4
D	4, 2	1, 1

Battle of the Sexes

	c	d
C	1, 1	3, 4
D	4, 3	2, 2

Staghunt

	c	d
C	4, 4	1, 3
D	3, 1	2, 2

Tricky Game

	a	b
A	0, 3	3, 2
B	1, 0	2, 1

Shapley's Game

	a	b	c
A	0, 0	0, 1	1, 0
B	1, 0	0, 0	0, 1
C	0, 1	1, 0	0, 0

Table 2.7: Six games of conflicting interest. In these matrix games, agents must offer and accept compromise to be successful when they associate with other successful learners.

2.5.3 Games of Conflicting Interest

In games of conflicting interest, agents do not necessarily agree on the solution that should be played. However, the agents need not compete. In fact, in these games, agents can offer and accept compromises to increase their average payoffs over time. Table 2.7 shows six games of conflicting interest. We now discuss these games since we we will be discussing the performance of learning algorithms in these games throughout this dissertation.

Prisoner's Dilemma The prisoner's dilemma is perhaps the most studied social dilemma (e.g., [2, 13]), as it appears to model many social situations. In the prisoner's dilemma, defection (D for player 1 and d for player 2) is each agent's dominant[13] action. The joint action in which both agents defect is pareto dominated (by the solution (C, c)). Both agents can increase their payoffs by coaxing the other agent to cooperate. To do so, an agent must (usually) be willing to cooperate (at least to some degree) in return. An n-agent, m-action version of this game is given by Stimpson [70, 71].

Chicken The game chicken has been shown to model the Cuban Missile Crisis [13] among other real-life situations. The game has two one-shot NEs ((C, d) and (D, c)).

[13]A *dominant action* produces a higher payoff (to the agent playing the action) than any other action regardless of the actions played by associates.

However, in a repeated game, either agent may be unwilling to repeatedly receive a payoff of 2 (its minimax value) when much more profitable solutions are available. Thus, in such cases, compromises can be reached, such as the rNE (C, c) (among others).

Battle of the Sexes This game models the stereotypical dilemma between husband and wife. They want to be together (hopefully) but do not always like doing the same things. This dilemma has been studied repeatedly (e.g., [13]) in the literature. Frequently, it is assumed that playing one of the one-shot NE solutions repeatedly, yielding payoffs of 4 to one player and 3 to the other, is acceptable behavior. However, an alternate (and possibly more desirable) solution is for the agents to alternate (take turns) receiving the high payoff (of 4).

Staghunt It has been argued that the staghunt models repeated social interactions more realistically than the prisoner's dilemma [69]. While it appears that the solution to the game is obvious (the solution (C, c) is best for both agents), fear of defection by the associating agent can lead to the NE (D, d), which gives each agent only a payoff of 2. This fear is increased if payoffs of 1 in this payoff matrix are lowered considerably, which case we will study in subsequent chapters.

Tricky Game This game, presented in [8] is an interesting game theoretically, although it is not likely to be seen in nature. In this game, $m_i = 1$ for each agent. The unique one-shot NE is for each agent to play randomly, resulting in an average payoff of 1.5. However, this one-shot NE is clearly pareto dominated by the solution (A, b) (among others), which is the solution suggested by Littman [53]). Agents can offer and accept compromises (as well as issue threats) to ensure that this solution (or some other closely related solution) is played.

Shapley's Game This game (which is studied in, among other places, [33]) has often been used to show that various learning algorithms do not converge. The game has a unique one-shot NE in which all agents play randomly. This NE yields a payoff of $\frac{1}{3}$ to each agent. However, if the agents alternate between "winning" (receiving a payoff of 1) and "losing" (receiving a payoff of 0), they each receive an average payoff of $\frac{1}{2}$. This pareto efficient strategy (which can be an rNE) is ignored in the literature.

2.6 Knowledge and Information

We mentioned previously that artificial agents should be able to learn given the knowledge afforded by their environments. When environments afford more information, agents should be able to use the extra information to learn more successfully (i.e., learn faster and learn better solutions). All of the games we have mentioned in this chapter can be played in environments that afford various amounts of knowledge to an agent. Below, we list the various categories of information (in order of increasing sophistication) that may be available to an agent in these (and other)

games. Knowledge of each category can be available to an agent in various degrees (i.e., probabilistically).

- **An agent's own actions.** Knowledge of the actions that one can take is a basic requirement of agency. An agent must be able to determine what it can do in order for it to make a choice.

- **An agent's own payoffs.** We also consider being able to observe the payoffs that one receives as a basic knowledge requirement for learning. However, information in this category may be available to an agent in various degrees. For example, an agent may sometimes know many consequences of various actions prior to acting in its environment, or it may need to learn these consequences through experimentation.

- **Existence of other agents.** Knowledge of whether other agents are acting on the same environment can dramatically increase an agents ability to learn.

- **Associates' actions.** Knowledge of the actions that other agents in the environment take can make the learning problem much easier in many situations. Information of this type may also be observable in varying degrees.

- **Associates' payoffs.** Being able to observe the payoffs that associates receive allows the interactions between agents to be much more complicated. For example, if an agent knows (to some degree) the payoffs that other agents receive, it can make intelligent threats to induce those agents to act in different ways.

- **Associates' internals.** Knowing (or being able to learn) the internal state and processes of associates can be very useful to an agent. However, it is not always practical, particularly if one's associates are complex. Information such as the learning processes employed by associates, learning rates, etc. fit into this category.

Note that when an agent does not know the payoffs and actions of other agents, offering and accepting beneficial compromises becomes increasingly difficult. Nevertheless, successful learners should learn to do so.

2.7 Discussion

The most influential approaches to multiagent learning to date have advocated that learning agents should seek to learn to play best-response strategies. Typically, this has been done by seeking to learn an equilibrium strategy based on the current episode (i.e., a myopic best response). Nevertheless, it has been shown that these (myopic) best-response approaches do not always produce adequate payoffs over time in games between learning agents [54]. Examples of games in which (myopic) best

response strategies demonstrate undesirable characteristics include social dilemmas, such as the iterated prisoner's dilemma, chicken, public goods games, etc. We advocate that learning algorithms must be developed that can learn to offer and accept profitable compromises (when beneficial) in these games. Furthermore, agents should learn to protect themselves from being exploited.

Chapter 3

Learning Near Pareto Efficient Solutions with Minimal Knowledge Requirements Using Satisficing

3.1 Introduction

Many applications to which multiagent learning can be applied require agents to cooperate with and adapt to each other on an ongoing basis. Thus, inherently, these applications of multiagent learning involve repeated play. This suggests that in many applications, the Nash equilibrium perspective should be replaced by a Nash bargaining perspective [59, 60], since learning a one-shot best-response (locally optimal) solution is not always desirable when agents interact repeatedly. It is, therefore, important that multiagent learning algorithms be developed that learn bargaining-like solutions. This entails that the algorithms learn to play pareto efficient or near pareto efficient solutions high percentages of the time when it is feasible, and have a solid fallback position when it is not. Such an approach also dictates that solutions be *fair*, meaning (loosely) that all agents receive relatively high payoffs.

Many real-world applications also require agents to learn in non-stationary environments where only partial information about the world and/or other agents is available. Thus, it is desirable for multiagent learning algorithms to learn effectively when minimal information is available to the agent.

In this chapter, we discuss multiagent learning in repeated general-sum matrix games with these two goals in mind: a) learning near pareto efficient solutions and b) requiring minimal knowledge/information about the environment. Specifically, we focus on learning near pareto efficient solutions when only knowledge of one's own actions and payoffs are available. Thus, we assume that an agent cannot observe the actions and payoffs of other agents.

While many multiagent learning algorithms have been developed, one of the more promising algorithms for learning pareto efficient solutions is a satisficing learning algorithm developed by Karandikar *et al.* [46]. Karandikar *et al.* analyzed the algorithm in a set of two-agent repeated play matrix games and found that, in self play, it converges to a pareto efficient (and cooperative) solution with probability close to unity. This algorithm was modified slightly and analyzed in a multiagent social dilemma (MASD) by Stimpson and Goodrich [72, 71, 70]. Again, they found that, when certain conditions are met, the satisficing algorithm (in self play) converges to

a pareto efficient solution with probability close to one in the MASD.

In this chapter, we analyze the algorithm presented by Stimpson and Goodrich (which we call the S-Algorithm) in the context of general-sum matrix games. We show that the S-Algorithm converges to a near pareto efficient solution (in self play) provided that certain conditions are met. We show that these conditions can be met in most repeated general-sum games (and all 2-agent general-sum games). Additionally, we present an empirical study showing the behavior of the S-Algorithm (in self play) in a variety of general-sum matrix games. In this study, we compare the performance of the S-Algorithm with the performance of two other well-analyzed learning algorithms (WoLF-PHC [9] and Q-learning [76]).

The structure of this chapter is as follows. In Section 3.2, we provide some background information. This includes a list of terms and definitions that we will use throughout this chapter, as well as a discussion of related work. We will then review previous work on aspiration-based satisficing learning in Section 3.3. In Section 3.4, we show that the S-Algorithm learns (in self play) to play near pareto efficient solutions in most general-sum matrix games with arbitrarily high probability, provided that certain initial conditions are met. We present an empirical study of the S-Algorithm in Section 3.5. In the last section, we summarize the chapter and provide further commentary on the S-Algorithm.

3.2 Background

In this section, we give some background information relevant to this chapter. First, we give terms and definitions that we will use throughout the chapter. We follow this with a discussion of some related work.

3.2.1 Terms and Definitions

In this chapter, we use the same notation related to strategies and payoffs that was introduced in the previous chapter. We now define additional terms and notation. We begin by revisiting the notion of pareto efficiency.

Definition 3.1. *(Pareto Dominated) A solution $\boldsymbol{\pi}$ is strictly pareto dominated if there exists a joint action $\mathbf{a} \in A$ for which $r_i(\mathbf{a}) > r_i(\boldsymbol{\pi})$ for all i and weakly pareto dominated if there exists a joint action $\mathbf{a} \in A$ for which $r_i(\mathbf{a}) \geq r_i(\boldsymbol{\pi})$ for all i.*

Definition 3.2. *(Pareto Efficient) A solution $\boldsymbol{\pi}$ is weakly pareto efficient (PE) if it is not strictly pareto dominated and strictly PE if it is not weakly pareto dominated. Unless specified, the former (i.e. weakly PE) terminology is implied.*

While pareto efficiency can be a goal of an agent who is involved in repeated play, it is sometimes difficult and impractical to guarantee. We relax this goal slightly by seeking near pareto efficiency.

	a	b	c
A	$1, 4$	$1.3, 2.8$	$2, 2$
B	$3, 3$	$0.6, 0.4$	$2.3, 0.9$
C	$4.5, 1$	$0.2, 2$	$3.2, 0.3$

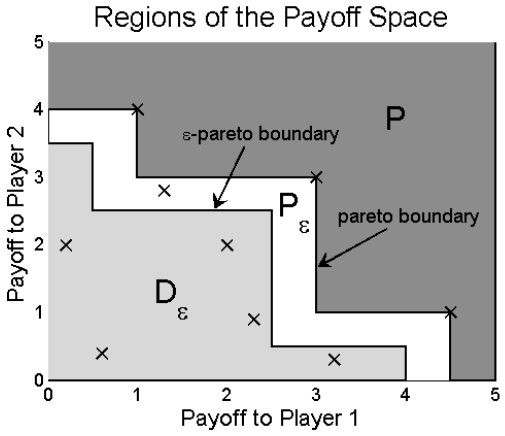


Figure 3.1: Diagram (shown at right) of the regions $\mathbb{P}$, $\mathbb{P}_\varepsilon$ (with $\varepsilon = 0.5$), and $\mathbb{D}_\varepsilon$ in the payoff space $\mathbb{R}^2$ for the 3×3 matrix game (shown at left).

Definition 3.3. *(ε-Pareto Dominated) A solution $\boldsymbol{\pi}$ is strictly ε-pareto dominated if there exists a joint action $\mathbf{a} \in A$ for which $r_i(\boldsymbol{a}) > r_i(\boldsymbol{\pi}) + \varepsilon$ for all i and weakly ε-pareto dominated if there exists a joint action $\mathbf{a} \in A$ for which $r_i(\mathbf{a}) \geq r_i(\boldsymbol{\pi}) + \varepsilon$ for all i.*

Definition 3.4. *(ε-Pareto Efficient) A solution $\boldsymbol{\pi}$ is weakly ε-pareto efficient (ε-PE) if it is not strictly ε-pareto dominated and strictly ε-PE if it is not weakly ε-pareto dominated. The former (i.e. weakly ε-PE) terminology is implied unless specified otherwise.*

We now give names to various regions of the reward space of n agents, some of which are shown (graphically) in Figure 3.1 for an arbitrary 3×3 matrix game (shown at left) with $\varepsilon = 0.5$. The $\times$'s depict the joint payoffs of the pure strategy solutions of the game. Let $\mathbb{R}^n$ denote the real-valued reward space. Let $\mathbb{D}$ be the subset of $\mathbb{R}^n$ that is weakly pareto dominated. Let $\mathbb{P} = \mathbb{R}^n - \mathbb{D}$. That is, $\mathbb{P}$ is the subset of $\mathbb{R}^n$ that is strictly PE. Also, let $\mathbb{D}_\varepsilon$ be the subset of $\mathbb{R}^n$ that is strictly ε-pareto dominated. Finally, let $\mathbb{P}_\varepsilon = \mathbb{D} - \mathbb{D}_\varepsilon$.

The *pareto boundary* divides set $\mathbb{D}$ from set $\mathbb{P}$ in the payoff space $\mathbb{R}^n$. If $\mathbf{v} \in \mathbb{D}$, then $\mathbf{v}$ is said to be below the pareto boundary. If $\mathbf{v} \in \mathbb{P}$, $\mathbf{v}$ is said to be above the pareto boundary. Likewise, the ε-pareto boundary divides set $\mathbb{D}_\varepsilon$ from set $\mathbb{P}_\varepsilon$. If $\mathbf{v} \in \mathbb{D}_\varepsilon$ or $\mathbf{v} \in \mathbb{P}_\varepsilon$, then $\mathbf{v}$ is said to be below or above the ε-pareto boundary, respectively.

Also, we say that the pure strategy solution $\mathbf{a} \in \mathbb{X}$, where $\mathbb{X}$ is some region of the payoff space, if $(r_1(\mathbf{a}), \ldots, r_n(\mathbf{a})) \in \mathbb{X}$.

It is also important to understand the way we use terms associated with satisficing. Let α_i^t be agent i's aspiration level at time t, and let $\boldsymbol{\alpha}^t = (\alpha_1^t, \ldots, \alpha_n^t)$ be the joint

aspirations of the agents at time t. A solution $\boldsymbol{\pi}$ is *satisficing* to agent i if $r_i(\boldsymbol{\pi}) \geq \alpha_i^t$. Likewise, an agent is *satisfied* if the most recently played joint action was satisficing to it. A solution $\boldsymbol{\pi}$ is *mutually satisficing* at time t if $r_i(\boldsymbol{\pi}) \geq \alpha_i^t$ for all i. Let $S_i(\alpha_i)$ be the set of pure strategy solutions that are satisficing to agent i given its aspiration level α_i. Then, we have the *mutually satisficing set* $S(\boldsymbol{\alpha}) = S_1(\alpha_1) \cap S_2(\alpha_2) \cap \cdots \cap S_n(\alpha_n)$.

3.2.2 Related Work

In this chapter, we discuss learning near pareto efficient solutions (in self play). Few learning algorithms have had this as a goal in the context of general-sum games, particularly when the actions and payoffs of other agents are not observable. We refer the reader to Section 2.2.2 for a further discussion on learning pareto efficient solutions in general-sum games.

Many algorithms have been developed which assume that the actions and payoffs of other agents are not available. In particular, many no-regret algorithms have this property (e.g., [31, 8, 43]), as well as WoLF-PHC [9], and standard (independent-action) Q-learners [76]. We note that each of these algorithms tends to play myopically in repeated matrix games, particularly in games in which no one-shot NE is pareto efficient.

This section discusses an aspiration-based satisficing learning algorithm. Satisficing, which is a term meaning good enough, has its roots in work by Simon on bounded rationality [68]. An alternate form of satisficing (using dichotomous attributes) is presented by Stirling and associates [74, 73, 58, 38], which is based somewhat on Levi's work on suspending judgment [48, 49]. A combination of Q-learning and dichotomous satisficing for single agent problems is manifest in Goodrich's Satisficing Q-learning algorithm [37].

3.3 Aspiration-based Satisficing Learning

In this section, we review two versions of the aspiration-based satisficing learning algorithm. First, we review the work of Karandikar *et al.* [46]. Second, we review the work of Stimpson and Goodrich [70, 72, 71].

3.3.1 Karandikar *et al.* (1998)

Karandikar *et al.* [46] studied the behavior of a satisficing algorithm in the set of 2×2 matrix games depicted in Figure 3.2. The authors imposed the following constraints on the values of the matrix: $\sigma > \delta > 0$ and $0 \leq \theta \neq \delta$. When $\theta > \sigma$ the game is a prisoner's dilemma.[1] Also, when $\theta = 0$, the game is a pure coordination game, where the joint actions (C, C) and (D, D) are both one-shot NEs.

[1]Note that this does not constitute a prisoner's dilemma as defined by Axelrod [2]. Axelrod imposed the additional constraint that $\sigma > \frac{\theta}{2}$.

	C	D
C	σ, σ	$0, \theta$
D	$\theta, 0$	δ, δ

Figure 3.2: 2×2 matrix game studied by Karandikar *et al.* The following constraints on the values of the matrix where imposed: $\sigma > \delta > 0$ and $0 \leq \theta \neq \delta$.

Repeat
(a) Select an action a_i^t according to the following rule:
if $(r_i^{t-1} < \alpha_i^{t-1})$ then with probability $1 - p$
$a_i^t \neq a_i^{t-1}$ (switch actions)
otherwise
$a_i^t = a_i^{t-1}$
(b) Receive reward r_i^t and update aspiration level:
With probability η
$\alpha_i^{t+1} = g(\alpha_i^t)$
Otherwise
$\alpha_i^{t+1} \leftarrow \lambda \alpha_i^t + (1 - \lambda) r_i^t$

Algorithm 3.1: The satisficing algorithm for agent i as presented by Karandikar *et al.*

The Algorithm

The satisficing learning algorithm as presented by Karandikar *et al.* is given in Algorithm 3.1. In short, an agent repeats its action of the previous episode whenever it is satisfied. Otherwise, the agent is disappointed and it switches its action with probability $1 - p$ (thus, it repeats the action of the previous episode with probability p). The probability p is based on the degree of the agent's *disappointment*, which is given by $\alpha_i^t - r_i^t$. The authors specify that p should be bounded away from 0, meaning that an agent continues its same action with at least some probability $\tilde{p} > 0$ (thus, $p \geq \tilde{p}$). The probability p is called the agent's *inertia*.

After both agents select an action, the resulting reward r_i^t is observed and the aspiration level (α_i^t) is either perturbed according to the distribution $g(\alpha_i^t)$[2] or moved toward r_i^t using the convex combination

$$\alpha_i^{t+1} \leftarrow \lambda \alpha_i^t + (1 - \lambda) r_i^t \tag{3.1}$$

[2]Karandikar *et al.* did not discuss the distribution $g(\alpha_i^t)$ in great detail other than specifying that aspiration trembles could occur on either side of α_i^t. Also, aspirations should remain in the region of *feasible payoffs*. In the next chapter, we will analyze how this distribution affects the long-term performance of the algorithm.

While most learning algorithms found in the literature have focused on learning the value of an action, Karandikar *et al.*'s algorithm focuses instead on learning an appropriate aspiration level. We will show in Chapter 5 that these approaches are indeed quite similar. However, the aspiration-based approach offers many important insights into the multiagent learning problem that are not immediately obvious (nor typically used) in action-based learning approaches.

Results

Karandikar *et al.* considered the situation in which two satisficing agents (with identical learning rates) play the set of matrix games defined in Figure 3.2. The behavior of these two agents can be described by a Markov process $\mathcal{P}^\eta$ over the state space, which can be identified as the set $E \equiv A \times \mathbb{R}^n$. A related process is the process $\mathcal{P}$, which describes the joint behavior of the agents in the absence of trembles. We refer to $\mathcal{P}^\eta$ as the *trembled process* and $\mathcal{P}$ as the *untrembled process*.

Let $s^t = (\mathbf{a}^{t-1}, \boldsymbol{\alpha}^{t-1})$ denote the state of the process at time t[3]. A *pure strategy state* is a state in $\alpha_i^{t-1} = r_i^{t-1}(\mathbf{a}^{t-1})$. Thus, if the untrembled process $\mathcal{P}$ is in a pure strategy state at time T, then $s^t = s^{t+1}$ for all $t > T$ thereafter. Karandikar *et al.* showed that, in the set of games depicted in Figure 3.2, $\mathcal{P}$ converges to some pure strategy state regardless of the initial state s^0.

The authors also showed that the trembled process $\mathcal{P}^\eta$ converges to a unique invariant distribution, regardless of the initial state of the algorithm. Additionally, they showed (and this is their main result) that the limit invariant distribution places almost all of its weight on the (C, C) pure strategy state, provided that the learning rate λ is sufficiently close to unity.

Intuition behind this result is depicted by Figure 3.3. Because of aspiration trembles (among other things), the joint aspirations will eventually move into the shaded region of the payoff space, which is the region in which only the (C, C) solution is mutually satisficing. Since agents repeat their actions when they are satisfied, the longer that aspirations remain in this region, the more likely the agent is to enter the (C, C) pure strategy state. Since λ controls how quickly aspirations change over time, λ close to unity means that the agent will spend most of its time in the (C, C) pure strategy state.

The argument (and corresponding proof) showing that λ controls the minimum amount of time that joint aspirations will remain in a given region was also made by Stimpson *et al.* (as we will see). Since we will also be using this result, we now review it (in the form of a lemma). We note that since this lemma assumes the absence of aspiration trembles, it is applicable only to the untrembled process $\mathcal{P}$.

[3]Note that s^t is determined by the joint action and joint aspirations of the previous episode since agent i's behavior is dependent on the comparison between r_i^{t-1} and α_i^{t-1} (see Algorithm 3.1).

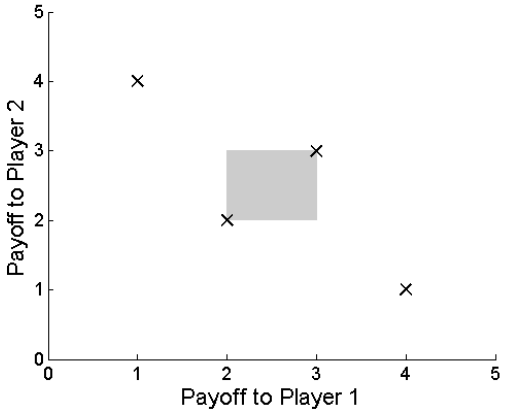


Figure 3.3: The payoff space of a particular prisoner's dilemma game. Once aspirations enter the shaded region, only mutual cooperation is mutually satisficing.

Let $T(\lambda, x)$ be the minimum number of episodes before a single agent's aspirations have deviated by x units (i.e., $|\alpha_i^{T(\lambda,x)+t} - \alpha_i^t| \geq x$) (given the learning rate λ).

Lemma 3.1. *As $\lambda \to 1$, $T(\lambda, x) \to \infty$ for all $x > 0$.*

Proof. It follows from the aspiration update rule given in Equation 3.1 that the maximum change in an agent's aspiration level in one episode is given by

$$|\alpha_i^{t+1} - \alpha_i^t| = (1-\lambda)|r_i^t - \alpha_i^t| \leq (1-\lambda)W \tag{3.2}$$

where W is the maximum difference between aspirations and payoffs, given by $\max(r_i^{\max} - \alpha_i^t, \alpha_i^t - r_i^{\min})$, where $r_i^{\max}$ and $r_i^{\min}$ are the maximum and minimum payoffs that an agent can receive on any given episode. Then it follows that

$$T(\lambda, x) = \left\lfloor \frac{x}{(1-\lambda)W} \right\rfloor \tag{3.3}$$

It is obvious from Equation (3.3) that $T(\lambda, x) \to \infty$ as $\lambda \to 1$. □

In the infinitely repeated games in question, the (C, C) outcome is as much as one can expect to receive (at least when limited to pure strategy outcomes) when associating with another learner. This is because both agents can *fallback* to the defection action (its minimax solution) if its associate tries to do better by defecting. However, aspiration-based satisficing learning algorithms is the only class of learning algorithms that we know of that learns mutual cooperation in self play of the iterated prisoner's dilemma when the actions and payoffs of other agents are unknown.

Karanikar *et al.* noted that the ability of this algorithm to learn mutual cooperation in these games comes with a price. While many (action-based) learning algorithms seek to learn a best response when associating with static (non-learning) agents, the satisficing learning algorithm does not. Indeed, the satisficing learning algorithm typically does not learn a best response to a static mixed strategy (and in some circumstances, a static pure strategy). The desirable features of the satisficing learning algorithm (i.e., convergence to pareto efficient solutions in which all agents benefit) arise from the joint behavior of the agents.

3.3.2 Stimpson *et al.* (2003)

Stimpson and Goodrich extended the satisficing learning algorithm to n-agent, m-action matrix games, and analyzed the algorithm in a multiagent social dilemma (MASD). We now review the resulting algorithm (which we call the S-Algorithm), the MASD (as presented in [71]), and Stimpson *et al.*'s findings.

Algorithm

The S-Algorithm is shown in Algorithm 3.2. The algorithm is identical to the satisficing learning algorithm presented by Karandikar *et al.* except for three important differences. First, the S-Algorithm requires (for success) that initial aspirations be set high[4]. Second, the S-Algorithm does not explicitly incorporate inertia (although an element of inertia still remains since an action yielding a non-satisficing result will be repeated with probability $\frac{1}{A_i}$). This modification becomes necessary for extending the algorithm to m-action games. Third, no aspiration trembles are employed by the S-Algorithm.

To better understand the S-Algorithm, consider Figure 3.4, which shows the payoff space of a 3×3 matrix game (shown with $\varepsilon = 0.5$). The payoffs to player 1 are plotted as x-coordinates and the payoffs to player 2 are plotted as y-coordinates. In the figure, $\times$'s indicate ε-PE solutions, $+$'s indicate non-ε-PE solutions, and the circle (o) indicates the current joint aspirations of the agents. All solutions in the region marked by the gray rectangle (the *satisficing region*) are mutually satisficing. At the time shown in the figure, the only solution in the satisficing region is ε-PE. Once the solution in the satisficing region is played, it will be played in every episode thereafter. However, if it is not played by the agents, aspirations are likely to relax until the satisficing region contains a (pure strategy) solution that is not ε-PE, which would make convergence to a non-ε-PE solution possible. Thus, as in Karandikar's

[4]We note that setting initial aspirations appropriately requires that various values of the game matrix are known. Even though the game matrix is unknown (and, even, unobservable), an agent can observe its highest payoff (with high probability) during an exploration period prior to beginning learning if associates also adequately explore during this time period. This value can be used to initialize an agent's aspiration level.

1. Set initial aspirations (α_i^0) high

2. Repeat
   (a) Select an action a_i^t according to the following criteria:
   $$a_i^t \leftarrow \begin{cases} a_i^{t-1} & \text{if } (\alpha_i^{t-1} \leq r_i^{t-1}) \\ \text{rand}(A_i) & \text{otherwise} \end{cases}$$
   where rand(A_i) denotes a random selection from the set A_i.

   (b) Receive reward r_i^t and update aspiration level:
   $\alpha_i^{t+1} \leftarrow \lambda\alpha_i^t + (1-\lambda)r_i^t$

Algorithm 3.2: The S-Algorithm for agent i as presented by Stimpson and Goodrich.

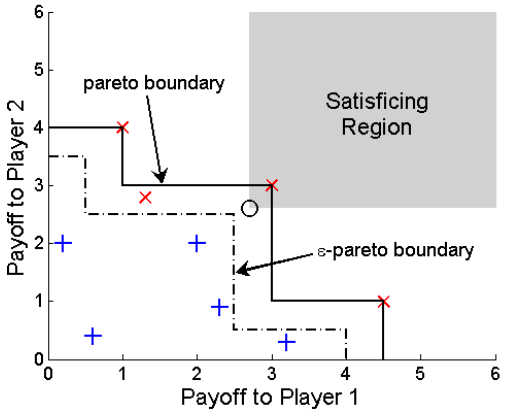


Figure 3.4: The payoff space of a 3×3 matrix game.

version of the algorithm, slower learning rates (i.e., high λ) make convergence to ε-PE solutions more likely.

A Multiagent Social Dilemma (MASD)

Stimpson and Goodrich analyzed the S-Algorithm in a multiagent social dilemma (MASD) [71, 70]. In each episode of the MASD, each of the n agents is faced with the decision of allocating M units of some discrete resource toward either its own goal or the group goal (i.e., a common goal shared by the n agents). Let a_i^t be the amount contributed by agent i toward the group goal at time t (and thus agent i contributes $M - a_i^t$ toward its own selfish goal). For each agent, there are $M+1$ possible actions ($a_i^t \in \{0, 1, \ldots, M\}$). Let each agent's payoff be represented as a linear combination

of the amount the group contributes to the group goal and the amount the agent contributes to its own goal. Let k_{G_i} be agent i's weighting of the group goal and let k_{S_i} be agent i's weighting of its own selfish goal. Then $r_i^t(\mathbf{a}^t)$ (agent i's payoff at time t when the joint action $\mathbf{a}^t$ is played) is given by

$$r_i^t(\mathbf{a}^t) = k_{G_i}\left[\sum_{j=1}^{n} a_j^t\right] + k_{S_i}(M - a_i^t) \tag{3.4}$$

If we suppose that only the relative (not absolute) utilities are important (and assume that all agents use the same weights k_G and k_S), the number of parameters can be reduced by letting $k_G = \frac{1}{nM}$ and $k_S = \frac{k}{M}$, where k is some positive constant[5]. A resulting relative payoff of $r_i^t(\mathbf{a}^t)$ can be used, given by

$$r_i^t(\mathbf{a}^t) = \frac{\left[\frac{1}{n}\sum_{j=1}^{n} a_j^t\right] - ka_i^t}{M(1-k)} \tag{3.5}$$

Figure 3.5 shows the payoff space of a two-agent MASD with $k = 0.6$, $M = 3$, and $n = 2$. The solution marked with the o is the solution in which both agents allocate all their resources to the group goal. The solution marked $*$ is the solution corresponding to the event in which each agent allocates all of its resources to its own individual goal. The two solutions marked +'s correspond to the events in which one agent gives all its resources to the group good, and the other contributes all its resources to its own individual goal. These four specially marked solutions correspond to the payoff space of a prisoner's dilemma, and the convex hull of these four solutions is equivalent to the convex hull of the MASD[6]. The other solutions of this game are marked with $\times$'s in the figure.

Results

Most of the work of Stimpson *et al.* focuses on the MASD. However, their initial work [72] gives an empirical study of the (slightly modified) S-Algorithm in the set of matrix games defined by Karandikar. The study analyzes how different learning rates and initial parameter settings affect the performance of the S-Algorithm in these games. Perhaps the most insightful of their findings is that the probability of mutual cooperation is not only dependent on the learning rate λ, but it is also dependent on how initial aspirations are set. They concluded that initial aspirations should be set high.

Stimpson *et al.* also showed that Q-learners and basic belief-based agents do not perform well in the MASD. They showed that the behavior of basic belief-based agents

[5]The game is really only interesting when $\frac{1}{n} < k < 1$.

[6]This insight shows that playing the classical iterated prisoner's dilemma is essentially a MASD (with $n = 2$) if the state of the agents is based on the previous M joint actions of the agents (when such information is observable).

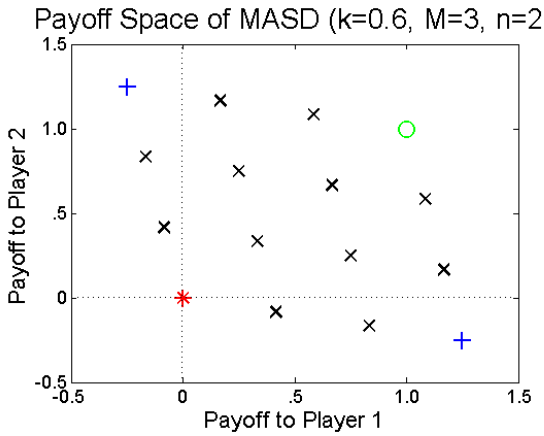


Figure 3.5: The payoff space of a 4×4 MASD ($k = 0.6$, $M = 3$, $n = 2$).

is essentially random in this game. On the other hand, if initial aspirations are high enough (i.e., above the pareto boundary) and if learning rates are slow enough, then the S-Algorithm learns to play pareto efficient solutions with high probability in the MASD (in self play). Additionally, if initial aspirations and learning rates are similar among the agents, then the pareto efficient solution that is most likely to be played is the Nash bargaining solution (the solution corresponding to the event in which all agents allocate all of their resources to the group goal).

Central to multiagent learning theory is the idea of security, or bounded loss [60]. Such is the concept behind the minimax value, etc. In the MASD, Stimpson *et al.* showed that the S-Algorithm also learns to play the minimax solution (defect, or contribute all resources to the individual goal) with high probability (dependent on λ) when associating with defecting agents. We note, however, that this result does not apply to games in which the minimax solution is a mixed strategy.

3.4 The S-Algorithm in General-Sum Matrix Games (Theoretical Results)

The works of Karandikar *et al.* and Stimpson *et al.* show in two different (yet related) contexts that aspiration-based satisficing learning results (in self play) in the agents learning to play a pareto efficient solution (with high probability). These results raise several questions. Does the S-Algorithm learn to play ε-pareto efficient solutions in all general-sum matrix games? If so, what conditions bring about this result? If not so, in what set of games does this result hold?

We answer these questions in this section. Before doing so, however, we analyze the untrembled process $\mathcal{P}$ in greater detail.

3.4.1 The Untrembled Process and Critical Events in General-Sum Games

Since the S-Algorithm has converged (learned) once it plays a mutually satisficing solution, it is essential that we have an understanding of the probability that mutually satisficing solutions will be played. That is, we must understand $Pr(\mathbf{a})$ for all $\mathbf{a} \in S(\boldsymbol{\alpha}^t)$, where $Pr(\mathbf{a})$ denotes the probability that the solution $\mathbf{a}$ will be played (given the current state of the process $\mathcal{P}$). When all agents act randomly, then $Pr(\mathbf{a}) = \frac{1}{|A|}$ for all $\mathbf{a}$. However, the S-Algorithm dictates that an agent act randomly only when it is not satisfied. Since it is common that at least one of the agents in the game will be satisfied (albeit temporarily), $Pr(\mathbf{a})$ is unknown in an unknown general-sum matrix game.

Fortunately, to show that the S-Algorithm can learn to play ε-PE solutions in general-sum matrix games (with high probability), we need only guarantee for some continuous subregion of the game's payoff space that a) only ε-PE solutions are mutually satisficing and that b) for each τ (a constant) episodes in which joint aspirations remain in this subregion, there is a positive probability that one such solution will be played. If we can guarantee that joint aspirations will remain in this subregion for $T \geq 1$ episodes, Lemma 3.1 shows us that T can be made arbitrarily large. This, in turn, means that the probability that the S-Algorithm will learn to play an ε-PE solution can be arbitrarily large.

Recall that the untrembled process $\mathcal{P}$ is defined over the continuous state space $A \times \mathbb{R}^n$. However, this state space be can partitioned into a finite[7] set of contiguous disjoint subregions by the observation that the probability that an action is selected at any given moment t is solely dependent on the previous joint action $\mathbf{a}^{t-1}$ and each agent i's satisficing set $S_i(\alpha_i^{t-1})$. Thus, when considering only action probabilities, the state s^t of $\mathcal{P}$ is given by $s^t = (\mathbf{a}^{t-1}, S_1(\alpha_1^{t-1}), \cdots, S_n(\alpha_n^{t-1}))$. Thus, $\mathcal{W}$ is called a *subregion*[8] of $\mathbb{R}^n$ if, for all $i \in I$, $S_i(\alpha_i)$ is constant for all $\boldsymbol{\alpha} \in \mathcal{W}$. In other words, a subregion is a subset of $\mathbb{R}^n$ in which all of the agents' satisficing sets are identical for all joint aspirations vectors in the region. Figure 3.6 demonstrates how satisficing sets partition $\mathbb{R}^n$ in this way for an arbitrary 2-agent game. Each rectangle in the figure is a separate subregion of $\mathbb{R}^n$.

From each joint aspiration vector $\boldsymbol{\alpha} \in \mathcal{W}$ (where $\mathcal{W}$ is a subregion of $\mathbb{R}^n$) the same set of critical events can occur. A *critical event* is a joint action (of a subset of the agents) that makes $Pr(\mathbf{a})$ become 0 for at least one $\mathbf{a} \in A$ for the remainder of the time that joint aspirations remain in the current subregion. In other words, a critical event is a joint action that reduces the number of solutions that will be played (for some period of time).

More formally, consider the example timeline shown in Figure 3.7. In the timeline, three different events are labeled. At time t_0, the joint aspirations $\boldsymbol{\alpha}$ enter the

[7]Finite if we assume a finite number of actions for each agent.

[8]Note the specific definition of the term *subregion* as it is used in this chapter.

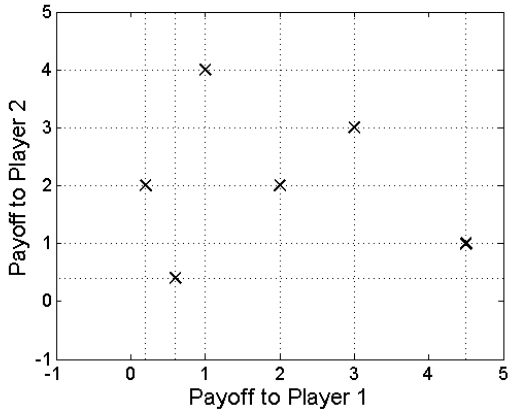


Figure 3.6: The payoff space $\mathbb{R}^n$ is partitioned into subregion in which all vectors in each subregion have a constant satisficing set S_i for all i.

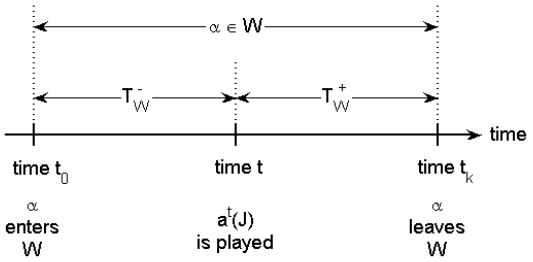


Figure 3.7: Timeline of the events surrounding a critical event.

subregion $\mathcal{W}$. At time t, the joint action $\mathbf{a}^t(J)$ is played, where $J \subseteq I$. At time t_k, the joint aspirations leave the subregion $\mathcal{W}$. Let $T^-_{\mathcal{W}}$ be the time interval $(t_0, t]$ and let $T^+_{\mathcal{W}}$ the time interval $(t, t_k]$ (as shown in the figure).

Definition 3.5. *(Critical Event)* $\mathbf{a}^t(J)$*, where* $J \subseteq I$*, is a critical event if, for some* $\mathbf{v} \in A$*, there exists some* $\zeta \in T^-_{\mathcal{W}}$ *such that* $Pr(\mathbf{a}^\zeta = \mathbf{v}) > 0$ *and* $Pr(\mathbf{a}^\tau = \mathbf{v}) = 0$ *for all* $\tau \in T^+_{\mathcal{W}}$*.*

We say that a critical event *excludes the solution* $\mathbf{v}$ if there exists some $\zeta \in T^-_{\mathcal{W}}$ such that $Pr(\mathbf{a}^\zeta = \mathbf{v}) > 0$ and $Pr(\mathbf{a}^\tau = \mathbf{v}) = 0$ for all $\tau \in T^+_{\mathcal{W}}$. Thus, a critical event is a joint action that excludes at least one solution $\mathbf{v} \in A$. If a solution is not excluded, then it is called a *non-excluded solution.* Also, we say that *critical event* $\mathbf{k}$ *excludes critical event* $\mathbf{d}$ if critical event $\mathbf{k}$ makes it impossible for critical event $\mathbf{d}$ to be played while aspirations remain in the current subregion.

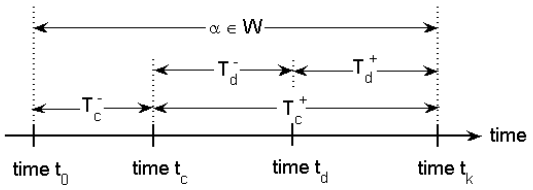


Figure 3.8: Timeline of the events surrounding two critical events in the same subregion.

Notice that the definition of a critical event does not discuss the effects of the critical event after the joint aspirations leave the current subregion. However, the effects of some critical events continue after joint aspirations have left the current subregion. If the effects of a critical event are permanent (i.e., the critical event makes it so that a solution will not be played for all $\tau > t$), then the critical event is said to be *stable*. Otherwise, the critical event is said to be *unstable*.

Multiple critical events can occur while joint aspirations remain in the same subregion, although no more than one critical event can occur on any given episode. Consider the timeline shown in Figure 3.8. In the figure, a critical event first occurs at time t_c, meaning that at least one solution $\mathbf{v} \in A$ is excluded by the joint action taken at time t_c until aspirations leave the current subregion. Note that in the Figure 3.8, the time intervals T_c^- and T_c^+ have replaced T_W^- and T_W^+ in Figure 3.7. A second critical event occurs at time t_d. We say that the critical event at time t_d *refines* the critical event at time t_c since it excludes at least one additional solution from being played for the remainder of the time that the joint aspirations remain in $\mathcal{W}$.

Three types of critical events occur with the S-Algorithm. They are *selection* critical events, *security* critical events, and *preclusion* critical events. We describe each type of critical event in the discussions that follow.

Selection Critical Events

Selection critical events are demonstrated by the game example shown in Figure 3.9. If the joint action (A, a) is played during the time in which joint aspirations are in the shaded subregion shown in the figure, then both agents will become satisfied (since they both receive a payoff at least as great as their aspiration level). Thus, they will play this joint action in each episode thereafter with probability 1, and the other three joint actions will not be played thereafter. Observe that any subregion to the left and down of the shaded subregion also contains this same critical event (but they may also contain other possible critical events as well).

	a	b
A	$(4, 4)$	$(1, 3)$
B	$(3, 1)$	$(2, 2)$

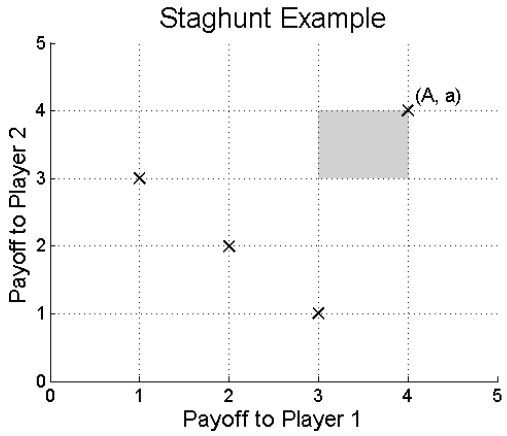


Figure 3.9: Example situation in which a selection critical event can occur. The joint action (A, a) is a selection critical event when joint aspirations are somewhere in the shaded subregion.

Formally, a selection critical event is a mutually satisficing solution. If a selection critical event is ε-PE, then the S-Algorithm has learned to play an ε-PE solution. On the other hand, if a selection critical event is not ε-PE, the S-Algorithm has learned to play a non-ε-PE solution. Note that all selection critical events are stable.

Security Critical Events

A second type of critical event, the security critical event, is illustrated by a second example, which is depicted in Figure 3.10. In this game, if joint aspirations are in the shaded subregion (or if the row player's aspiration level is 3 or less), a security critical event occurs when player 1 (the row player) plays A. Once player 1 plays A, it will continue to do so from then on (and, thus, the joint actions (B, a) and (B, b) will have zero probability of being played from then on), while player 2 will play randomly until its aspiration level falls to 2 (or below). At this point, the joint aspiration vector will be in the subregion just below the shaded subregion, and a second critical event (a selection critical event) will eventually refine the first critical event when player 2 plays the action a.

The joint action (B, a) in Figure 3.10 is a selection critical event if it is played when joint aspirations are in the shaded subregion. If this critical event is played before the security critical event (in which player 1 plays A), then the security critical event is excluded. Similarly, once the security critical event (in this case) is played, the selection critical event (B, a) is excluded.

In the example just given, the security critical event in which player 1 plays A is stable, meaning that even after the joint aspirations leave the subregion in which the critical event occurred, the solutions (B, a) and (B, b) continue to be excluded.

	a	b
A	$(3, 2)$	$(3, 1)$
B	$(4, 3)$	$(1, 4)$

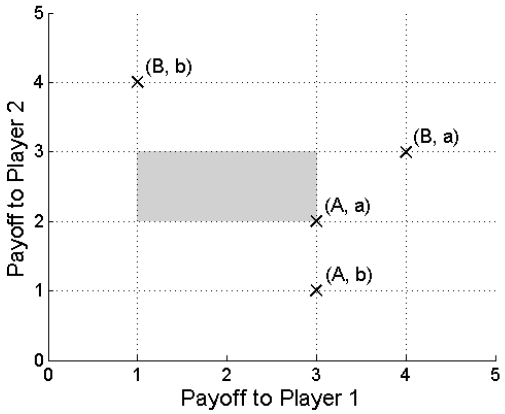


Figure 3.10: Situation in which a security critical event can occur. When joint aspirations are in the shaded subregion (or anytime player 1's aspirations are 3 or less), then a security critical event occurs when player 1 plays A.

	a	b
A	$(3.5, 2)$	$(3, 1)$
B	$(4, 3)$	$(1, 4)$

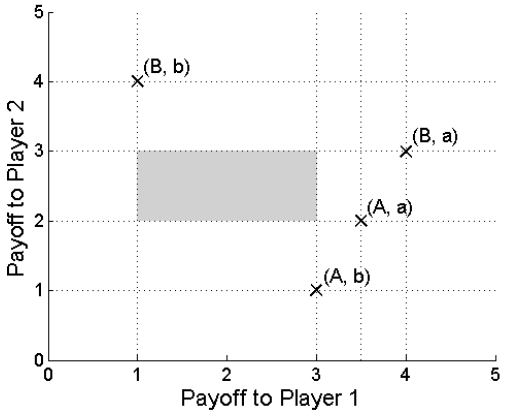


Figure 3.11: Situation in which an unstable security critical event can occur.

However, security critical events can be unstable as well. To see this, consider the matrix game shown in Figure 3.11. This game is identical to the game shown in Figure 3.10 except that player 1 receives a payoff of 3.5 rather than 3 when the joint action (A, b) is played. In this game, the security critical event still occurs when player 1 plays A when joint aspirations are in the shaded subregion. However, if aspirations enter a subregion to the right of the shaded subregion (i.e., $\alpha_1^t > 3$), then player 1 will become dissatisfied if player 2 plays b. Thus, player 1 would again play randomly, so the solutions (B, a) and (B, b) would no longer be excluded.

We are now ready to formally define security critical events.

Let $r_i^s(a_i) = \min_{\mathbf{a}_{-i} \in A_{-i}} r_i(a_i, \mathbf{a}_{-i})$ be the *security* of action a_i to agent i. In words, agent i will never receive a payoff lower than $r_i^s(a_i)$ when it plays action a_i. Thus, if the joint aspirations are in a subregion such that $\alpha_i^t \leq r_i^s(a_i)$, then a_i is a security critical event since, once it is played, agent i will continue to play it until joint aspirations leave the current subregion. This critical event excludes every entry in the payoff matrix except for those entries corresponding to $(a_i, \cdot)$.

The definition just given does not consider the case in which a critical event (of some kind) has already excluded some solutions. In such cases, agent i's security for taking action a_i might differ from its original value. Let $\mathcal{E}^t$ be the set of non-excluded solutions at time t. Then, agent i's security for taking action a_i at time t is

$$r_i^s(a_i|\mathcal{E}^t) = \min_{\mathbf{a}_{-i} \in A_{-i}, (a_i, \mathbf{a}_{-i}) \in \mathcal{E}^t} r_i(a_i, \mathbf{a}_{-i}) \tag{3.6}$$

Thus, a_i is a critical event when $r_i^s(a_i|\mathcal{E}^t) \geq \alpha_i^t$. We continue to use the notation $r_i^s(a_i)$ to denote the special case in which $\mathcal{E}^t = A$. Thus, $r_i^s(a_i) = r_i^s(a_i|A)$.

It is possible that any of an agent's actions can be security critical events (under certain conditions). However, when $\mathcal{E}^t = A$ and initial aspirations are set high (as called for by the S-Algorithm shown in Algorithm 3.2), the first possible security critical event that each agent can play is its *minimax action*. Let $\pi_i^P = \text{argmax}_{a_i \in A_i} r_i^s(a_i)$ be agent i's minimax action (over the pure strategy space), and let m_i^P be agent i's *minimax value* over the pure strategy space given by

$$m_i^P = \max_{a_i \in A_i} r_i^s(a_i) = \max_{a_i \in A_i} \min_{a_{-i} \in A_{-i}} r_i(a_i, a_{-i}) \tag{3.7}$$

Thus, if agent i's minimax action is not a security critical event in a given subregion, then neither are any of its other actions.

We note that, in the case of a critical event, the minimax action and value for each agent can change (in accordance to the security $r_i^s(a_i|\mathcal{E}^t)$ of each action $a_i \in A_i$). Thus, more generally, let $\pi_i^P(\mathcal{E}^t) = \text{argmax}_{a_i \in A_i} r_i^t(a_i|\mathcal{E}^t)$ and let $m_i^P(\mathcal{E}^t) = r_i^s(\pi_i^P|\mathcal{E}^t)$. In words, the minimax strategy and value (over the pure strategy space) are calculated by considering only those solutions that are not currently excluded. Note that $m_i^P(\mathcal{E}^t) \geq m_i^P$.

	a	b	c
A	$(0, 3)$	$(3, 0.5)$	$(3, 0)$
B	$(0.5, 3)$	$(4, 4)$	$(1.5, 3)$
C	$(3, 1.5)$	$(3, 1)$	$(1, 3)$

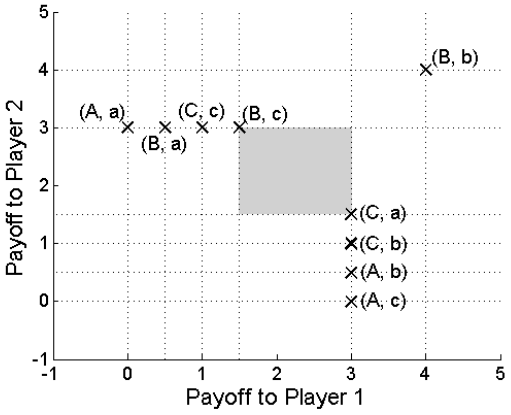


Figure 3.12: Situation in which a preclusion critical event can occur. In this case, if joint aspirations are in the shaded subregion, then a cyclic critical event can occur which makes it impossible for the joint action (B, b) to be played thereafter.

Preclusion Critical Events

The game shown in Figure 3.12 illustrates a preclusion critical event. When joint aspirations are in the shaded subregion in this game, then all joint actions except the joint action (B, b) are preclusion critical events since they make it impossible for the joint action (B, b) to be played thereafter (while aspiration remain in the shaded subregion). For example, if the joint action (C, a) is played, then player 1 will continue to play action C until player 2 plays action c, at which point player 2 will be satisfied (receiving a payoff of 3). Thus, player 2 will continue to play c until player 1 plays action A. At this point, player 1 (who again receives a payoff of 3) will be satisfied until player 2 plays action a, meaning that player 2 will be satisfied until player 1 has played action C. This cycle will continue to repeat (at least while the joint aspirations remain in the shaded subregion). At no time during the cycle can the action (B, b) ever be played.

The defining property of preclusion critical events is that, after a preclusion critical event, no agent is satisfied with all non-excluded solutions. This is not true of selection and security critical events. However, as in the case of all critical events, after a preclusion critical event, each non-excluded solution does satisfy at least one of the agents (which is what causes the effective reduction in the size of the payoff matrix).

Let J be any subset of I in which $|J| \geq 2$. The agents in J can induce/play a preclusion critical event if a) there exists a vector of thresholds $T^{\text{prec}} = (T_j^{\text{prec}}, ..., T_{j-1+|J|}^{\text{prec}})$ such that $\alpha_j^t \leq T_j^{\text{prec}}$ for all $j \in J$ and b) all non-excluded solutions (as a result of the critical event being played) satisfy at least one agent $j \in J$ (i.e., if the critical event occurs at time t, then for all $\mathbf{a} \in \mathcal{E}^{t+1}$, $r_j(\mathbf{a}) \geq T_j^{\text{prec}}$ for some $j \in J$). If these two

	c	d
C	$(1, 3)$	$(3, 2)$
D	$(4, 1)$	$(4, 1)$

Table 3.1: Game containing a refinable preclusion critical event.

conditions do not hold, then no preclusion critical event can occur.

Preclusion critical events are classified as either unrefinable or refinable. An *unrefinable preclusion critical event* (of which the game shown in Figure 3.12 is an example) excludes all other critical events for the duration of the time that joint aspirations are in the current subregion. Thus, all non-excluded solutions (as a result of the critical event being played) satisfy at least one agent $j \in J$, *but not all* $j \in J$. As a result, the play of the agents cycles.

As an example, consider, again, the game depicted in Figure 3.12. In this game, the thresholds $T_1^{\text{prec}} = 3$ and $T_2^{\text{prec}} = 3$ delineate the regions of support for a preclusion critical event. Thus, when $\alpha_i^t \leq 3$ for $i = 1, 2$, then a preclusion critical event can occur in which the solution (B, b) is excluded. Note that all non-excluded solutions gives one of the agents (but not both) at least its threshold value.

As in security critical events, the threshold T_j^{prec} can change as the set of non-excluded solutions ($\mathcal{E}^t$) changes. Thus, we denote the threshold as $T_j^{\text{prec}}(\mathcal{E}^t)$ for the general case.

A *refinable preclusion critical event* can be refined by a selection or security critical event while aspirations remain in the current subregion. An example of a refinable preclusion critical event is given by the matrix game shown in Table 3.1. In this game, suppose that $\boldsymbol{\alpha}^{t-1} = (3.5, 2.9)$, and that the joint action played at time t is (C, c). In this case, player 2 (the column player) would be satisfied, whereas player 1 would not be. Thus, player 2 would repeat the action c until player 1 played D, which is a stable security critical event, resulting in player 1 playing D thereafter. Thus, the preclusion critical event (C, c) excludes the solution (C, d).

Note that in this example, the preclusion critical event (C, c) is only a critical event because of the security critical event that follows. But this is not true of all refinable critical events. For example, if aspirations are in the subregion to the left of the shaded subregion in the game shown in Figure 3.12, then the same preclusion critical event that was discussed previously (in this game) is a refinable preclusion critical event since the mutually satisficing solution (B, c) (which is a selection critical event) can refine the preclusion critical event. Refinable preclusion critical events occur only when a) there exists at least one solution that is mutually satisficing (i.e., $S(\boldsymbol{\alpha}) \neq \emptyset$) or b) $\alpha_i^t \leq m_i^P(\mathcal{E}^t)$ for some i.

Refinable preclusion critical events are governed by the same kind of thresholds T_j^{prec} as unrefinable preclusion critical events. For example, in the game shown in

Table 3.1, $T_1^{\text{prec}} = 4$ (player 1's minimax value) and $T_2^{\text{prec}} = 3$. This example illustrates the fact that if $T_j^{\text{prec}} \leq m_j^P$ for some j, then the preclusion critical event is refinable by a security critical event. A preclusion critical event is also refinable by a selection critical event if a non-excluded solution has a payoff that exceeds the threshold of each agent.

Before proceeding, we present one more important property of preclusion critical events. We give this property in the form of a lemma.

Lemma 3.2. *A preclusion critical event cannot exclude a security critical event.*

Proof. The proof follows directly from the defining characteristic of preclusion critical events. That is, after a preclusion critical event, no agent is satisfied with all non-excluded solutions. This means that each agent has a positive probability of becoming dissatisfied (over a constant number of episodes), at which point it will act randomly. Thus, all of its actions have a positive probability of being played (and, thus, they cannot be excluded by a preclusion critical event). □

We are almost ready to present our main results about the S-Algorithm. Before doing so, however, we formalize restrictions on initial aspirations.

3.4.2 Assumptions about Initial Aspirations

The first step of the S-Algorithm (shown in Algorithm 3.2) specifies that initial aspirations $\boldsymbol{\alpha}^0$ should be set *high*. As this term is somewhat vague, we now make some clarifications. Perhaps the simplest method for initializing aspiration levels to high values would be for each player to observe through experimentation (before learning) the various payoffs it could receive. After each agent completes its observation period, it could then set its aspiration level (α_i^0) to the highest payoff it has seen (up to that point).

While this methodology for choosing initial aspirations is practical (assuming the length of the observation period for each agent is similar), it is overly restrictive. Instead we impose two restrictions on initial aspirations. First, $\boldsymbol{\alpha}^0 \in \mathbb{P} \cup \mathbb{P}_{\frac{\varepsilon}{2}}$, meaning that initial aspirations will not be below the $\frac{\varepsilon}{2}$-pareto boundary. Second, since an agent should not be content with payoffs below its minimax value (over the pure strategy space) $\alpha_i^0 > m_i^P - \frac{\varepsilon}{2}$ for all i (where m_i^P is defined as in Equation 3.7). The term $\frac{\varepsilon}{2}$ is subtracted from m_i^P to ensure that initial joint aspirations can be in $\mathbb{P}_{\frac{\varepsilon}{2}}$). These two rules define the region $\mathbb{P}_0$.

3.4.3 Main Theoretical Results

We are now ready to answer whether (or when) the S-Algorithm learns to play ε-PE solutions with high probability in self play. We show that the S-Algorithm does learn to play ε-PE solutions with high probability determined by λ in most

general-sum matrix games [9] when certain conditions (which we define shortly) are met.

Let $\mathcal{R}$ be the set of subregions of $\mathbb{R}^n$ defined by the satisficing sets of the agents. Let $\mathcal{R}^P \subseteq \mathcal{R}$ be the set of subregions such that if $\mathcal{W} \in \mathcal{R}^P$, then $\forall \boldsymbol{\alpha} \in \mathcal{W}$, $S(\boldsymbol{\alpha}) \neq \emptyset$ and $\forall \mathbf{a} \in S(\boldsymbol{\alpha})$, $\mathbf{a} \in \mathbb{P}_\varepsilon$.[10] In words, $\mathcal{R}^P$ is the set of subregions in which only ε-PE (pure strategy) solutions can possibly be mutually satisficing and at least one solution is mutually satisficing.

Note that $\mathcal{R}^P$ contains (at the least) all subregions that overlap (i.e., share space with) $\mathbb{P}_\varepsilon$. Thus, we know that $\mathcal{R}^P$ is not empty. In many games, $\mathcal{R}^P$ contains other subregions as well. For example, in the game staghunt shown in Figure 3.9, $\mathcal{R}^P$ contains three additional subregions: $(x \in (1,2], y \in (2,3])$, $(x \in (2,3], y \in (1,2])$, and $(x \in (2,3], y \in (2,3])$.

However, it is not enough for $\mathcal{R}^P$ to be non-empty. In order to show that that the S-Algorithm will converge to ε-PE solutions with probability dependent on λ, we must show that joint aspirations will enter at least one of the subregions $\mathcal{W} \in \mathcal{R}^P$ in which no critical event (or sequence of critical events) can exclude all mutually satisficing solutions. If this happens, we say that the *conditions of pareto efficiency* have been met.

Let $\mathcal{R}^N \subseteq \mathcal{R}^P$ be the set of subregions in which no critical event (or sequence of critical events) can exclude all mutually satisficing (ε-PE) solutions. Further, for simplicity, let $\mathcal{R}^N$ consist only of those subregions of $\mathcal{R}^P$ that share space with $\mathbb{P}_{\frac{\varepsilon}{2}}$.

We first show that when $\mathcal{R}^N \neq \emptyset$, then it is possible to learn ε-PE solutions with high probability. We then give circumstances under which $\mathcal{R}^N \neq \emptyset$.

Theorem 3.1. *If $\mathcal{R}^N \neq \emptyset$, then there exists a non-empty set of vectors $\mathbb{V} \in \mathbb{P}_0$ such that if $\boldsymbol{\alpha}^0 \in \mathbb{V}$, then the S-Algorithm learns (in self play) to play ε-PE solutions with arbitrarily high probability dependent on λ.*

Proof. To show that the S-Algorithm learns to play an ε-PE solution with high probability, we need only show that, with high probability, a mutually satisficing ε-PE solution will be played once (since once one of these solutions is played, it will be played from then on).

Let $\mathcal{W}$ be a subregion of $\mathcal{R}^N$. Let $\acute{\mathcal{W}}$ be the interior of $\mathcal{W}$. Then let $\mathbb{V} \equiv \acute{\mathcal{W}} \cap \mathbb{P}_{\frac{\varepsilon}{2}}$.[11] Note that $\mathbb{V}$ is non-empty since $\mathcal{R}^N$ (which consists of subregions in $\mathcal{R}^P$ that share space with $\mathbb{P}_\varepsilon$) is assumed non-empty (in the statement of the theorem).

[9] The phrase *most general-sum games* does not entirely make sense, since there are an infinite number of general-sum games. However, each matrix game specifies, for each agent, a preference ordering over solutions, and there are a finite set of preference orderings over solutions (if we assume $|A|$ is finite). Thus, we use the phrase *most general-sum games* to indicate that most sets of preference orderings specify situations in which the S-Algorithms learns to play pareto efficiency solutions with probability dependent on λ when certain conditions are met. We note that all the games that have been studied in the literature (that we are aware of) have such preference orderings.

[10] Recall that $S(\boldsymbol{\alpha})$ is the mutually satisficing set for the joint aspirations $\boldsymbol{\alpha}$.

[11] $\mathbb{V}$ can be (and usually is) larger than $\acute{\mathcal{W}} \cap \mathbb{P}_{\frac{\varepsilon}{2}}$, but this suffices for the proof.

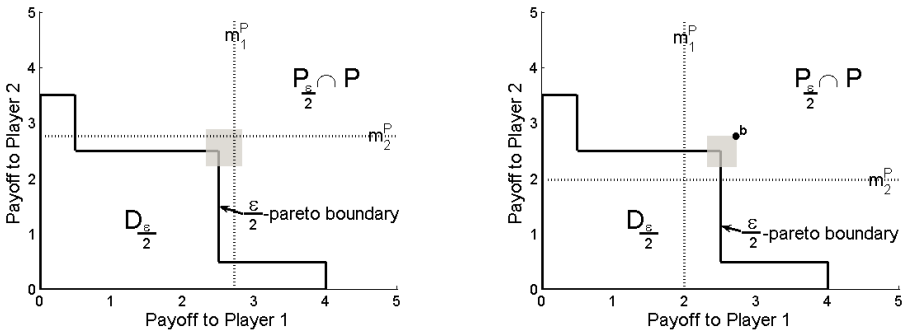


Figure 3.13: Examples showing a subregion that is in $\mathcal{R}^N$ in games with no preclusion critical events.

If $\boldsymbol{\alpha}^0 \in \mathbb{V}$, joint aspirations must remain in $\mathcal{W}$ for a non-zero length of time (i.e., number of episodes), the length of which is dependent on λ. Let $T(\lambda)$ denote this minimum length of time. Since $\boldsymbol{\alpha}^t \in \mathcal{W}$ (where t is one of the $T(\lambda)$ episodes), there exists at least one ε-PE solution that is mutually satisficing ($S(\boldsymbol{\alpha}^t) \neq \emptyset$). Thus, there exists some positive probability $p \in (0, 1]$ that some $\mathbf{a} \in S(\boldsymbol{\alpha}^t)$ will be played during some constant number of episodes τ (since $\boldsymbol{\alpha}^t \in W \in \mathcal{R}^N$). This means that the probability that some $\mathbf{b} \in S(\boldsymbol{\alpha}^t)$ will *not* be played over τ episodes is $1 - p$, so the probability that some $\mathbf{b} \in S(\boldsymbol{\alpha}^t)$ will *not* be played during $T(\lambda)$ episodes is $q = (1-p)^{\frac{T(\lambda)}{\tau}}$. Since $T(\lambda) \to \infty$ as $\lambda \to 1$ (see Lemma 3.1), $q \to 0$ as $\lambda \to 1$. □

This theorem shows that *if* $\mathcal{R}^N$ is non-empty, then there exists initial joint aspirations for which the S-Algorithm will converge to an ε-PE solution with high probability. However, we have not shown *when* $\mathcal{R}^N$ is non-empty. In the theorems that follow, we show that $\mathcal{R}^N$ is guaranteed to be non-empty in most general-sum games. However, there exists a *small* class of games in which $\mathcal{R}^N$ appears to be empty[12]. We now give these results.

The first class of games that we consider is the large set of games that have no preclusion critical events.

Theorem 3.2. *$\mathcal{R}^N \neq \emptyset$ in games that have no preclusion critical events.*

Proof. In these games, we must show that there exists a subregion such that neither a selection critical event nor a security critical event can exclude all mutually satisficing ε-PE solutions.

[12]Again, since the number of general-sum matrix games in infinite, the number of games in this *small* class of games is also infinite. But there are a finite set of preference orderings over solutions (if we assume $|A|$ is finite). Thus, we use the term *small* in this case to indicate that there is only a small number of sets of preference orderings (for general-sum games) for which $\mathcal{R}^N$ appears to be empty.

For selection critical events, we note that when initial joint aspirations are in a subregion in $\mathcal{R}^P$, then only ε-PE selection critical events can occur. Thus, selection critical events cannot exclude all mutually satisficing ε-PE solutions when aspirations are in any subregion of $\mathcal{R}^P$. Thus, if only selection critical events can occur, then $\mathcal{R}^N \equiv \mathcal{R}^P$, which is non-empty.

We must now show that a security critical event cannot exclude all mutually satisficing ε-PE solutions in all subregions in $\mathcal{R}^P$. In this case, it is sufficient to only consider each agent's minimax action π_i^P as security critical events. This is because an agent's other actions can only be security critical events if its minimax action is a security critical event (see Section 3.4.1). Thus, we need consider only the security critical events π_i^P (for all i).

If $(m_1^P, \cdots, m_n^P) \in \mathbb{P}_{\frac{\varepsilon}{2}} \cup \mathbb{P}$, then the joint action $(\pi_1^m, \cdots, \pi_n^m)$ is $\frac{\varepsilon}{2}$-PE. Since no security critical event can exclude the solution $(\pi_1^m, \cdots, \pi_n^m)$ when $\boldsymbol{\alpha} = (m_1^P, \cdots, m_n^P)$, the subregion containing the point $(m_1^P, \cdots, m_n^P)$ is in $\mathcal{R}^P$. A hypothetical situation of this kind is shown in Figure 3.13(left). In this example, the shaded subregion is a subregion in $\mathcal{R}^P$.

In contrast, if $(m_1^P, \cdots, m_2^P) \notin \mathbb{P}_{\frac{\varepsilon}{2}} \cup \mathbb{P}$ (see Figure 3.13(right)), then there must be at least one PE solution $\mathbf{b} \in \mathbb{P}$ such that $r_i(\mathbf{b}) > m_i^P + \frac{\varepsilon}{2}$ for all i. Thus, $\mathbf{b}$ cannot be excluded by a security critical event and so the subregion containing the point $(r_1(\mathbf{b}), \cdots, r_n(\mathbf{b}))$ is in $\mathcal{R}^P$ (as illustrated in the figure).

Thus, there exists a subregion in $\mathcal{R}^P$ such that no selection or security critical events can exclude all mutually satisficing (and ε-PE) solutions. Thus, $\mathcal{R}^N \neq \emptyset$, since these games have no preclusion critical events. □

Most general-sum games do not have preclusion critical events, particularly unrefinable ones. However, for a complete treatment of general-sum matrix games, we must discuss games in which a preclusion critical event can occur for some set of joint aspirations $\boldsymbol{\alpha}$. Games with preclusion critical events have the high-level characteristics depicted in Figure 3.14(left). In the figure (which is depicted for a two agent game), the agents' minimax actions and preclusive thresholds define the (shaded) region in which no security or preclusion critical events can occur. This region is the set of points $p = (p_1, \ldots, p_n) \in \mathbb{R}^n$ such that $p_i > m_i^P$ for all i and, for all J in which $|J| \geq 2$, $p_j > T_j^{\text{prec}}$ for some $j \in J$. If any solution $\mathbf{a} \in \mathbb{P}_\varepsilon$ is in this region, then $\mathcal{R}^N$ is non-empty, since the subregion containing the solution $\mathbf{a}$ is in $\mathcal{R}^N$ (the darkly shaded region in Figure 3.14(right)). We give this result in the form of a theorem.

Theorem 3.3. *$\mathcal{R}^N \neq \emptyset$ in games in which there is a preclusion critical event if there exists a solution $\mathbf{a} \in \mathbb{P}_\varepsilon$ such that $r_i(\mathbf{a}) > m_i^P$ for all i and $r_j(\mathbf{a}) > T_j^{\text{prec}}$ for some $j \in J$ (for all possible sets $J \subseteq I$ in which $|J| \geq 2$).*

Proof. Let $\mathcal{W}$ be the subregion containing the solution $\mathbf{a}$. Since $\mathbf{a} \in \mathbb{P}_\varepsilon$, then $\mathcal{W} \in \mathcal{R}^P$. Suppose that $\boldsymbol{\alpha}^0 \in \mathcal{W}$. Then, a selection critical event cannot exclude $\mathbf{a}$ unless it is

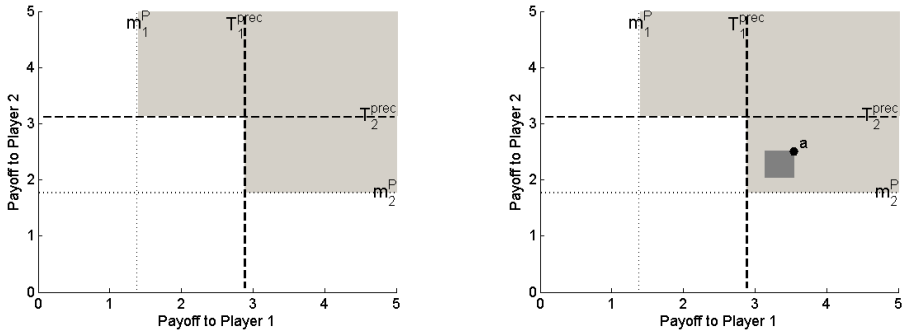


Figure 3.14: (Left) High level characteristics of games with preclusion critical events. (Right) When a solution $\mathbf{a} \in \mathbb{P}_\varepsilon$ is in the shaded region, then $\mathcal{R}^N$ is non-empty, since the darkly shaded subregion is in $\mathcal{R}^N$.

itself ε-PE. Also, a security critical event cannot exclude $\mathbf{a}$ since $r_i(\mathbf{a}) > m_i^P$ for all i. Lastly, a preclusion critical event (of any kind) cannot exclude $\mathbf{a}$ since $\alpha_j > T_j^{\text{prec}}$ for some $j \in J$ for all possible sets of J in which $|J| \geq 2$. Thus, $\mathcal{W} \in \mathcal{R}^N$, so $\mathcal{R}^N$ is non-empty. □

Note that this result also holds when $T_j^{\text{prec}} < m_j^P$ (although this is not depicted in Figure 3.14(left)).

When there is no solution in the shaded regions of Figure 3.14(left), then either a preclusion or security critical event can occur in all subregions of $\mathcal{R}^P$. However, we show that in all 2-agent general-sum matrix games, these critical events cannot exclude all ε-PE mutually satisficing solutions for all possible initial joint aspiration vectors. Thus, $\mathcal{R}^N$ is not empty in any 2-agent matrix game.

Theorem 3.4. *$\mathcal{R}^N \neq \emptyset$ in all 2-agent general-sum matrix games.*

Proof. If the game has no preclusion critical events, then Theorem 3.2 suffices. Additionally, if the game has a solution that satisfies the stipulations given in Theorem 3.3, then the proof of that theorem suffices. Hence, we need only consider games in which there does not exist a solution that satisfies the stipulations given in Theorem 3.3 (the shaded region of Figure 3.14(left)), but that does contain a preclusion critical event.

First, we consider unrefinable preclusion critical events. In these games, the points $(T_1^{\text{prec}}, m_2^P)$ and $(m_1^P, T_2^{\text{prec}})$ are both in $\mathbb{P}_\varepsilon$, otherwise the conditions stated in Theorem 3.3 would be satisfied. Additionally, we claim that there must be at least one solution $\mathbf{a} \in A$ that weakly dominates the point $(T_1^{\text{prec}}, m_2^P)$ (i.e., $r_1(\mathbf{a}) \geq T_1^{\text{prec}}$ and $r_2(\mathbf{a}) \geq m_2^P$) and at least one other solution $\mathbf{b} \in A$ that weakly dominates the point $(m_1^P, T_2^{\text{prec}})$ (i.e., $r_1(\mathbf{b}) \geq m_1^P$ and $r_2(\mathbf{b}) \geq T_2^{\text{prec}}$).

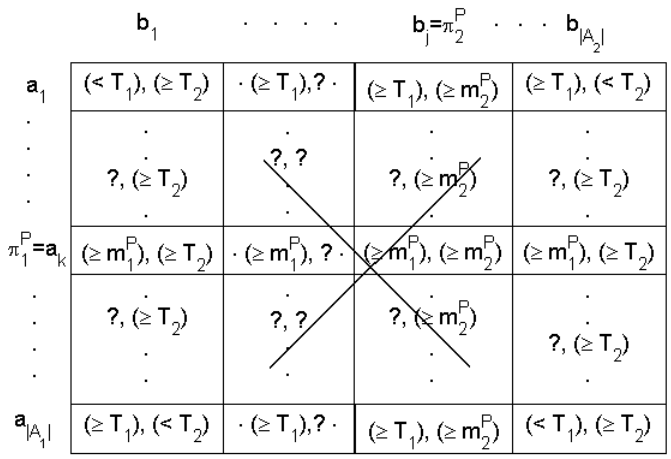


Figure 3.15: The structure of a payoff matrix of 2-agent games with an unrefinable (possibly) preclusion critical event. In the figure, $T_i = T_i^{\text{prec}}$.

To see this, consider Figure 3.15, which gives the general structure of the payoff matrix for all 2-player games of this kind[13]. As in all 2-agent payoff matrices, the rows in the figure (labeled a_1 to $a_{|A_1|}$) give the actions available to the row player and the columns (labeled b_1 through $b_{|A_2|}$) give the actions available to the column player. Thus, the joint action (a_1, b_1) gives player 1 (the row player) a payoff less than T_1^{prec} (*prec* is omitted for convenience) and it gives player 2 a payoff greater than or equal to T_2^{prec}. Dots in the figure indicate other rows and columns not depicted in the figure.

As in previous examples that we have given of unrefinable preclusion critical events (see Figure 3.12), the solutions in the center of the payoff matrix are excluded (as shown by the big $\times$ in Figure 3.15) by a preclusion critical event when $\alpha_j^t \leq T_j^{\text{prec}}$ for all $j \in J$. Thus, once the critical event is played, the agents then cycle through the solutions on the perimeter of the payoff matrix until another critical event occurs or one of the agent's aspiration levels exceeds its preclusive threshold. For example, if the solution $(a_{|A_1|}, b_1)$ is played, then player 1 will continue to play the action $a_{|A_1|}$ until player 2 plays $b_{|A_2|}$ (since $\alpha_1 < T_1$), at which point player 2 will play $b_{|A_2|}$ until player 1 plays a_1 (since $\alpha_2 < T_2$), etc.

In any event, observe that the preclusion critical event does not exclude at least one solution that weakly dominates the point $(m_1^P, T_2^{\text{prec}})$ (in the case of the figure, when player 1 plays π_1^P and player 2 plays b_1 or $b_{|A_2|}$). The same argument can be made for the point $(T_1^{\text{prec}}, m_2^P)$.

[13]Figure 3.15 gives the general structure of the payoff matrix of these games. Rows and columns can be shuffled. Additionally, more than two rows and two columns can be involved in the resulting cycle (caused by the preclusion critical event).

Thus, all solutions that weakly dominate these points are ε-PE solutions in 2-player games (unless, again, the payoff matrix satisfies Theorem 3.3, which means that the proof of that theorem suffices). Also, the subregions containing these solutions are in $\mathcal{R}^N$, provided that no refinable preclusion critical event can exclude all of these solutions. A refinable preclusion critical event cannot exclude all mutually satisficing (ε-PE) solutions in these subregions since a) these subregions are in $\mathcal{R}^P$ (thus, a non-ε-PE selection critical event cannot refine a preclusion critical event) and b) the only possible security critical events in these regions are the agents security actions (m_i^P) (or security critical events that yield nearly the same security to the agent), which cannot exclude all mutually satisficing solutions for joint aspirations in these subregions. □

We note that setting initial aspirations in these subregions of $\mathcal{R}^N$ may not be (and probably is not) in the best interest of at least one of the agents, as other aspiration levels could give it higher average payoffs. However, if aspirations are placed in these subregions, the S-Algorithm will learn to player ε-PE solutions in self play with high probability dependent on λ.

Unfortunately, it appears that the last theorem cannot be extended to ($n > 2$)-agent matrix games, since a preclusion critical event (played by a different set of agents) could theoretically exclude the solutions in the subregions defined in the proof. Thus, $\mathcal{R}^N$ can possibly be empty in some small class of games. Thus, in these games we cannot guarantee that the S-Algorithm will converge with high probability to ε-PE solutions.

On a positive note, it appears that these games (i.e., games in which we cannot guarantee that $\mathcal{R}^N$ is non-empty due to preclusion critical events) are very rare and uninteresting[14], as we know of no game studied in the literature that possesses these properties. The same can be said of any game with an unrefinable critical event (that we know of).

Regardless of whether there is a set of initial joint aspirations that causes the S-Algorithm to learn to play ε-PE solutions with high probability, S-Algorithm agents are protected by security critical events. Thus, an agent's average payoffs are likely bounded from below (in the limit) by the minimax value m_i^P. This is because the agent is likely (if λ is close enough to unity) to play π_i^P (a security critical event) before its aspirations drop far below m_i^P. This argument follows from Lemma 3.2, which says that a preclusion critical event cannot exclude a security critical event. Recall that Stimpson *et al.* showed this security property for the specific case of the MASD [71].

In the next section, we will demonstrate the individual average payoffs obtained by the S-Algorithm (in self play) in many theoretically important games. Before

[14]Uninteresting in the sense that an agent is not likely to encounter such games in practical situations.

doing so, however, we (briefly) discuss convergence of the S-Algorithm (again, in self play).

3.4.4 Convergence

Throughout this chapter, we have repeatedly discussed *convergence*. We now formally define the term (as we use it) and discuss the convergence of the S-Algorithm in general-sum games.

Definition 3.6. *(Convergence) A learning process has converged if and only if* $\pi_i^t = \pi_i^{t+1}$ *for all* $t > T$*, where* T *is some constant.*

The question arises whether or not the S-Algorithm converges in self play in all general-sum matrix games. While the S-Algorithm does converge with high probability when aspirations begin in a subregion of $\mathcal{R}^N$, it is not guaranteed to converge when initial aspirations vectors are not in a subregion of $\mathcal{R}^N$. Thus, if $\mathcal{R}^N$ is empty or agents do not set their aspirations "appropriately," then the S-Algorithm might not converge (in self play).

While there are many examples of such games (especially many games with unrefinable preclusion critical events), we present just one (additional) example. This game is shown in Figure 3.16. In this game, if joint aspirations are initialized above the pareto boundary, then aspirations will remain above the pareto boundary (i.e., $\alpha^{\mathbf{t}} \in \mathbb{P}$ for all t). Thus, no selection critical event can ever occur. In general, the joint aspiration vector will remain close to $(0.25, 0.25)$ (with high probability), which is a point in $\mathbb{P}$.

Observe, however, that if joint aspirations are chosen using the method suggested in Theorem 3.1, then initial joint aspirations will be in the subregion containing the point $(0, 0)$. This means that any solution is mutually satisficing (as well as ε-PE). Thus, the S-Algorithm will converge to an ε-PE solution with probability 1. However, note that convergence to PE solutions does not entail effective learning (in all cases), as, in this case, it requires that at least one of the agents receive an average payoff of 0. In Chapter 5, we present a learning algorithm that is able to alternate between "winning" and "losing" so that both agents receive an average payoff of 0.5 in this game.

3.5 The S-Algorithm in General-Sum Matrix Games (Empirical Results)

In the previous section, we showed that the S-Algorithm converges to ε-PE solutions in self play (with high probability) when the conditions of pareto efficiency are met. We showed that the conditions of pareto efficiency can be met in most general-sum matrix games. These conditions are easily identifiable when the complete payoff matrix is known. However, in this chapter, we have assumed that the game matrix

	a	b
A	$(0, 0)$	$(0, 1)$
B	$(1, 0)$	$(0, 0)$

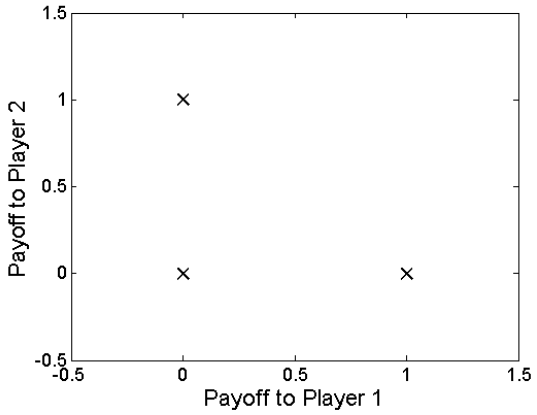


Figure 3.16: Example game in which the S-Algorithm does not converge.

is not known (and cannot even be constructed since the payoffs and actions of other agents are unknown).

That being said, we note that the conditions of pareto efficiency are not difficult to satisfy in most general-sum games if the agents takes a little time to estimate their highest possible payoff ($r_i^{\max}$). More importantly, the matrix games that have been studied extensively in the literature are in the class of games in which these conditions are easily met.

In this section, we present an empirical study of the S-Algorithm in a variety of theoretically important general-sum matrix games. In doing so, we compare the S-Algorithm in self play to two other algorithms: WoLF-PHC [9] and Q-learning [76]. Both of these algorithms (like the S-Algorithm) operate under the assumption that the actions and rewards of other agents (and, thus, the payoff matrix) are unknown. Unless stated otherwise, the parameter settings for the three algorithms (abbreviated S-Algorithm, WoLF, and QL respectively) are given in Table 3.2. For the S-Algorithm, initial aspirations where randomly selected in the region $\mathbb{P}_0$ with $\varepsilon = 0$[15].

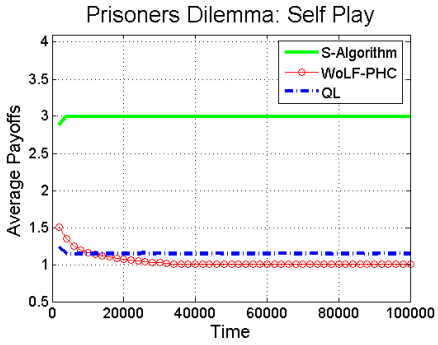
Figures 3.17-3.25 show the performance of the algorithms in the prisoners dilemma, MASD ($k = 0.6, n = 3, M = 5$), chicken, staghunt, Shapley's game, tricky game [8], battle of the sexes, and two other (unnamed) games. For each game, we show the average performance over time of the algorithms (over 50 trials) and a box and whisker plot (generated by MATLAB) showing the final (average) payoff distributions. These box and whisker plots show the median (line in middle of box), the median of the upper and lower quartiles (top and bottom of box), and extremes of the data (top and bottom of whiskers). Data points which fall outside of 1.5 multiplied by the

[15]We acknowledge that the agents do not know how to set their aspirations so that $\boldsymbol{\alpha} \in \mathbb{P}_0$ when the payoff matrix is unknown. However, in this empirical study, we assume that the agents do set their aspirations as such. In the next chapter, we consider the case in which the agents perturb their aspiration levels, which eliminates the need for this assumption.

Name	Parameter Values
S-Algorithm	$\alpha_{\mathbf{0}} \in \mathbb{P}_0$, $\lambda = 0.99$
WoLF-PHC	$\lambda = 1/(10 + 0.01\kappa_s^a)$, $\gamma = 0.95$, ϵ-greedy w/ $\epsilon = \max(0.2 - 0.0006t, 0)$
QL	$\gamma = 0$, $\lambda = 1/(10 + 0.01\kappa_s^a)$, ϵ-greedy w/ $\epsilon = 0.1$

Table 3.2: Learners and their parameter values. κ_s^a is the number of times that action a has been played in state s, λ is used for the learning rate to avoid confusion. QL uses a stateless Q-update.

	c	d
C	3, 3	0, 5
D	5, 0	1, 1



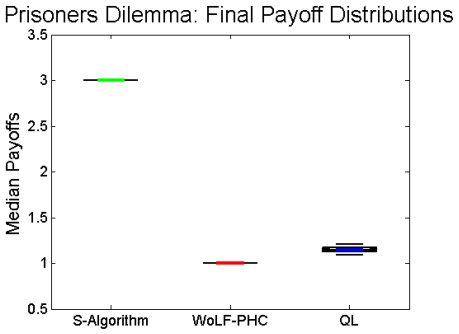


Figure 3.17: Results from 50 trials of the iterated prisoner's dilemma.

interquartile range are considered outliers, and are plotted as $*$'s. Notches in the boxes represent a robust estimate of the uncertainty about the medians for box-to-box comparison. Boxes whose notches do not overlap indicate that the medians of the two groups differ at a 95% confidence level. Tighter distributions are generally more desirable than distributions that vary widely.

The results as given in the figures are largely self explanatory. However, we comment on various aspects of each.

3.5.1 Prisoner's Dilemma

As has been shown by Karandikar *et al.* and Stimpson *et al.*, the S-Algorithm learns mutual cooperation in the prisoner's dilemma (see Figure 3.17). For the parameters (and initial aspirations) used, the S-Algorithm learned mutual cooperation in all cases. Both WoLF-PHC and QL learn without exception (in the 50 trials) to always defect. We note that QL's payoffs are slightly higher than the mutual defection payoff since it plays randomly with probability 0.1.

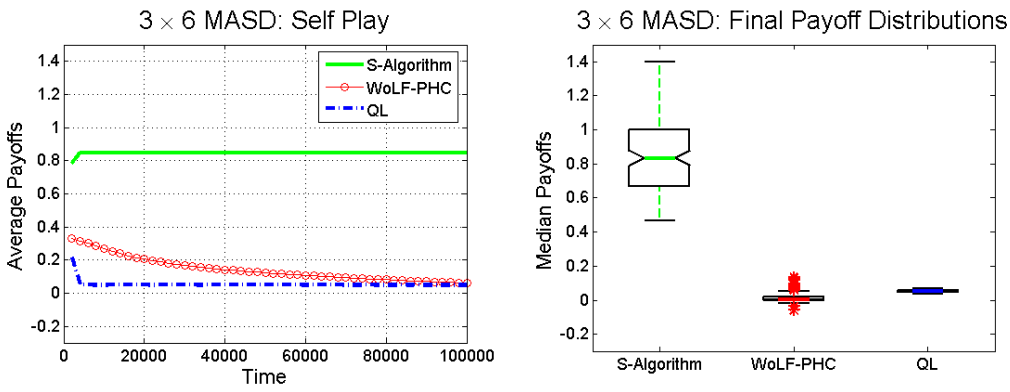


Figure 3.18: Results from 50 trials of a 3-agent, 6-action MASD (k=0.6). In this game, mutual cooperation yields a payoff of 1 to each agent and mutual defection produces a payoff of 0 to each agent.

3.5.2 MASD

We also tested the algorithms on a 3-agent, 6-action MASD (k=0.6) (see Figure 3.18). Both WoLF-PHC and QL learn to play selfishly (almost always completely selfishly), while the S-Algorithm generally learns to play cooperatively. Note, however, that full cooperation does not always result. The large variation in the data is due to changing selections of $\alpha^{\mathbf{0}}$. Again, we note that Stimpson *et al.* showed that the S-Algorithm generally learns mutual cooperation when initial aspirations are high and similar [70] (and λ is sufficiently high). Additionally, because of the large joint action space of this game, a higher λ is needed to make mutual cooperation more likely.

3.5.3 Chicken

Figure 3.19 shows the performance of the three algorithms in the game *chicken*. While the S-Algorithm learns without fail to always cooperate, the other two algorithms learn to either cooperate while the other defects, defect while the other cooperates, or to both cooperate. The result is that the average payoffs received by the S-Algorithm are significantly higher than those received by the other algorithms.

3.5.4 Staghunt

The performance of the three algorithms in the game staghunt is shown in Figure 3.20. In this game, the S-Algorithm distances itself from WoLF-PHC and QL, since its learns mutual cooperation (with one exception). WoLF-PHC and QL learn the safer (yet less profitable) strategy of defection. We note that QL's payoff is lower than WoLF-PHC's because of its continued exploration.

	c	d
C	$3, 3$	$2, 3.5$
D	$3.5, 2$	$1, 1$

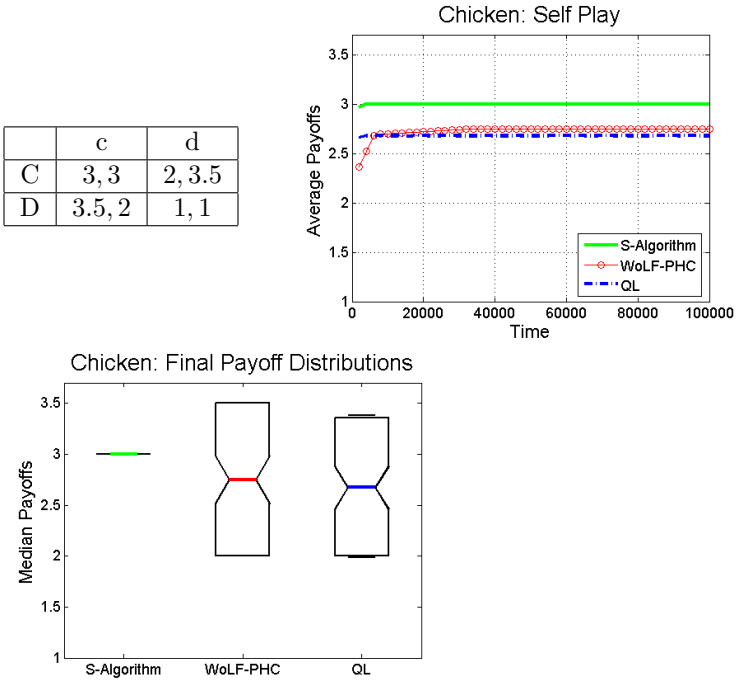


Figure 3.19: Results from 50 trials of the matrix game chicken.

	c	d
C	$4, 4$	$-5, 3$
D	$3, -5$	$2, 2$

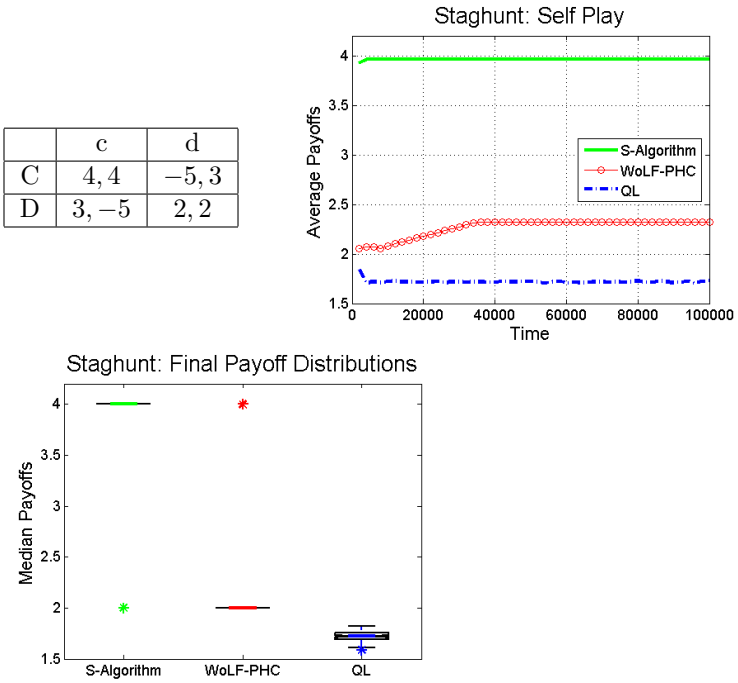


Figure 3.20: Results from 50 trials of the matrix game staghunt.

	a	b	c
A	0, 0	0, 1	1, 0
B	1, 0	0, 0	0, 1
C	0, 1	1, 0	0, 0

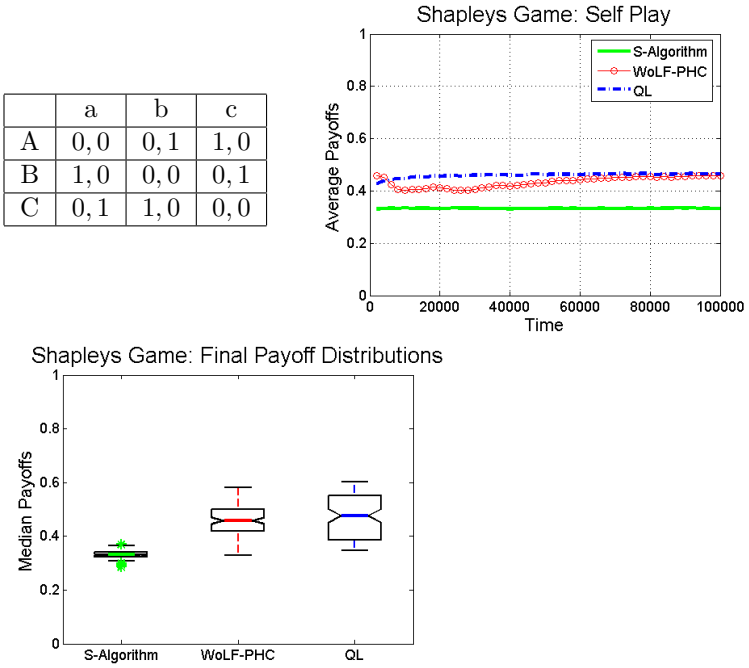


Figure 3.21: Results from 50 trials of Shapley's game.

3.5.5 Shapley's Game

The S-Algorithm does not perform well in Shapley's game, as shown in Figure 3.21. While WoLF-PHC and QL play strategies that yield average payoffs close to 0.5,[16] the S-Algorithm receives (in self play) an average payoff of $\frac{1}{3}$. This is not because the S-Algorithm does not play ε-PE solutions. Indeed, all payoffs in Shapley's game are ε-PE. Rather, the joint aspirations of the agents never fall below the pareto frontier (in pure strategy space). Thus, the S-Algorithm's behavior is essentially random (the one-shot NE). This behavior arises because the S-Algorithm focuses only on the current episode.

Note that if aspirations are set as described in Theorem 3.1, the S-Algorithm would converge. However, one of the agents would necessarily receive a payoff of 0 on each episode. In order for learning agents to perform effectively in Shapley's game, they must learn turn-taking strategies. The algorithm we present in Chapter 5 is able to do this successfully.

[16] Game theorists and others have repeatedly used Shapley's game to show the failure of learning algorithms to converge to an one-shot NE (see, for example, [33]). Interestingly, learning algorithms (even those that focus only on the current episode) tend to perform better than they would if they played the one-shot NE (see Figure 3.21 and [8]).

	a	b
A	0, 3	3, 2
B	1, 0	2, 1

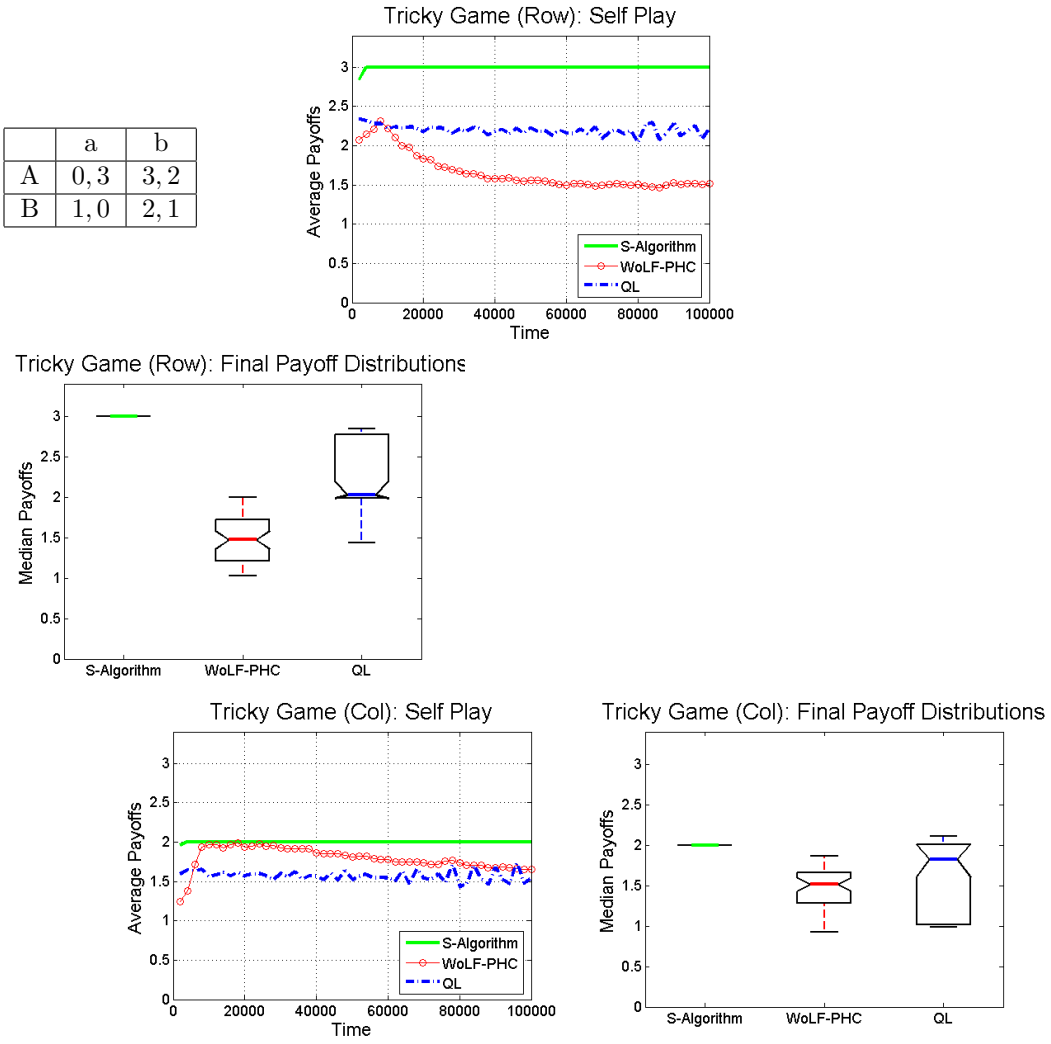
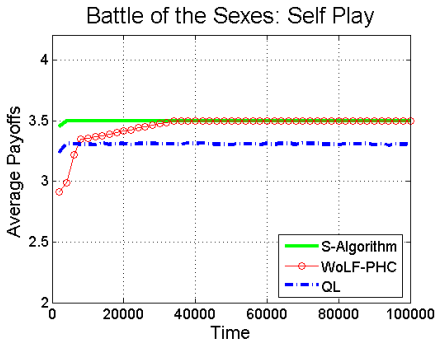


Figure 3.22: Results from 50 trials of Tricky Game. The performance of the row player is shown above and the performance of the column player is shown below.

3.5.6 Tricky Game

In Tricky game the S-Algorithm, again, outperforms the other two algorithms (see Figure 3.22), both as the row player and as the column player. In each of the 50 trials, the S-Algorithm learned the PE solution in which the row player receives a payoff of 3 and the column player receives a payoff of 2. The learned behaviors of both WoLF-PHC and QL vary more widely and typically yield a lower payoff to both the row and column players. Note that WoLF-PHC plays somewhat *cooperatively* in early rounds of the game. However, myopic best response approaches cannot sustain cooperation in this game. Thus, it eventually learns (as designed) to play the one-shot NE (which yields a payoff of 1.5 to each agent).

	c	d
C	$2, 2$	$4, 3$
D	$3, 4$	$1, 1$



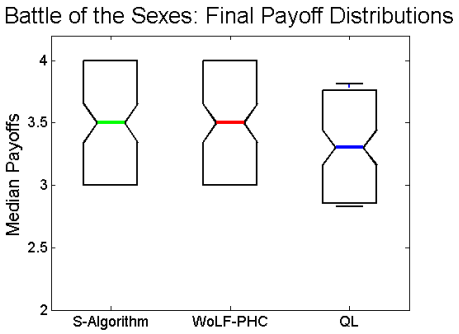


Figure 3.23: Results from 50 trials for a version of the matrix game Battle of the Sexes.

3.5.7 Battle of the Sexes

Results for the game Battle of the Sexes are shown in Figure 3.23. The performance of all the agents is essentially equivalent. Either the row player receives 4 and the column player 3, or vice versa. Both solutions are PE. Note that, again, QL's payoffs are slightly lower since it continues to act randomly with probability 0.1. Since the algorithms focus only on the current episode, they are unable to learn an alternate (but fairer) PE solution in which the agents alternate between receiving payoffs of 3 and 4.

3.5.8 Negative Result

In the previous games, the S-Algorithm learns to play PE solutions with high probability for all initial aspiration vectors in $\mathbb{P}_0$.[17] We now present a game in which many initial aspiration vectors (in $\mathbb{P}_0$) lead to the play of a stable security critical event that excludes all ε-PE solutions. The game is shown in Figure 3.24. This game has two (pure strategy) PE solutions, both of which occur when player 1 (the row player) plays B. However, if the row player's aspiration level drops below 3.5, then a stable security critical event occurs if the row player plays the action A.

The proof of Theorem 3.1 locates initial joint aspirations under which the S-Algorithm will converge (with high probability) to a ε-PE solution in this game. Specifically, if $\alpha^{\mathbf{0}} \approx (4, 3)$, then the S-Algorithm will converge to the PE solution (A, a) with high probability. However, the S-Algorithm is not guaranteed to learn to learn to play an ε-PE solution with high probability for all $\alpha^{\mathbf{0}} \in \mathbb{P}_0$, since joint aspirations might not pass through a subregion in $\mathcal{R}^N$ in this game.

[17] Again, we note that even in Shapley's game, the S-Algorithm learns to play PE solutions for all initial aspiration vectors in $\mathbb{P}_0$ even though it does not always converge, as all solutions are weakly PE. Additionally, its average payoffs are PE.

The performance of the S-Algorithm (as well as that of WoLF-PHC and QL) in this game is shown in Figure 3.24. Indeed, in most trials, the security critical event does occur before a mutually satisficing, PE solution is played, resulting (eventually) in convergence to the (A, a) solution. It is interesting to note that WoLF-PHC and QL appear to be even more susceptible than the S-Algorithm to this event, as they learn similar strategies (and never learn to play the PE solution). We note, that, as shown in the proof, the S-Algorithm converges with high probability to the PE solution when initial aspirations are initially set to $(3.99, 2.99)$. This result is shown in Figure 3.24 as *S-Alg (Fixed)*.

In Figure 3.25, we modify this game so that the security critical event is unstable. While this does not change the behavior of WoLF-PHC and QL, it does allow the S-Algorithm to learn to play the PE solution with higher probability. In fact, the S-Algorithm learns to play (B, b) (with high probability) for most initial aspiration vectors in this game. However, when the row player has relatively low initial aspiration levels compared to the column player, the security critical event excludes the PE solution long enough for convergence to a non-PE solution to become possible.

3.6 Summary, Discussion, and Future Work

In this chapter, we reviewed the satisficing learning algorithms presented by Karandikar *et al.* and Stimpson *et al.* We also analyzed the S-Algorithm (the algorithm presented by Stimpson) in general-sum matrix games. We showed that in almost all general-sum matrix games (and all 2-agent matrix games), the S-Algorithm learns (in self play) to play ε-PE solutions with high probability dependent on λ if the conditions of pareto efficiency are satisfied (see Section 3.4.3). However, since the payoffs and actions of other agents are not known (and the payoff matrix is not known), guaranteeing that these conditions are satisfied is impossible. But these conditions can be easily met when the game matrix is known.

Because it is difficult to determine how to set initial aspirations so that the conditions or pareto efficiency are satisfied in an arbitrary (unknown) game, we presented an empirical study which demonstrates that these conditions are easily met in a number of well-studied games. In many of these games, all initial aspiration vectors in the region $\mathbb{P}_0$ allow the conditions of pareto efficiency to be met. We also showed that even when the conditions or pareto efficiency are not satisfied, the S-Algorithm still performs as well (in self play) as two other well-studied (and typical) multiagent learning algorithms.

We emphasize that these results are for self play only. Future work could include analysis of the S-Algorithm when it associates with other kinds of learners. The S-Algorithm, however, does not promise to do well when associating with a large class of learning agents, as the success of the learning algorithm is closely tied to the joint behavior that emerges as a result of adherence to the satisficing principle [46].

	a	b
A	$3.5, 2$	$3.5, 1$
B	$1, 4$	$4, 3$

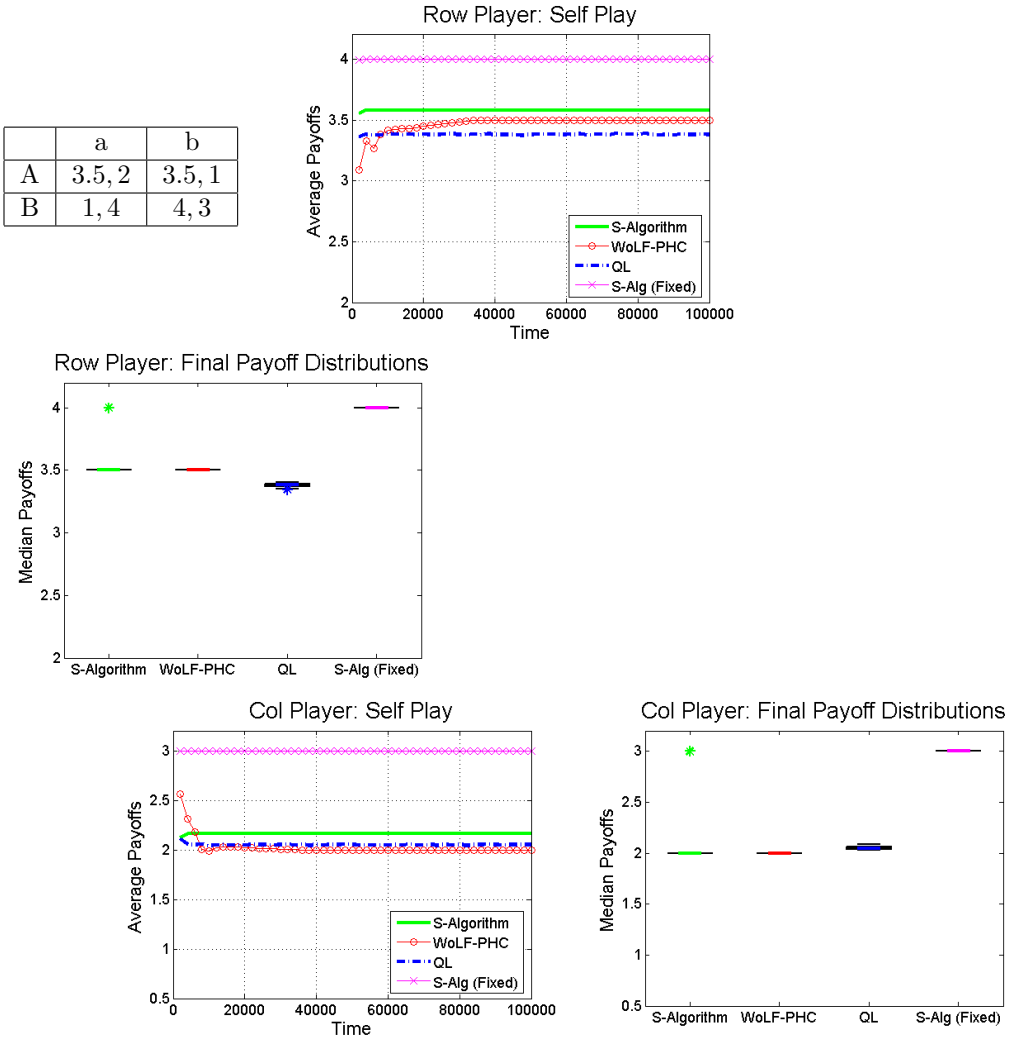


Figure 3.24: Results from 50 trials of the game shown at left. The performance of the row player is shown above and the performance of the column player is shown below.

	c	d
C	$3.6, 2$	$3.5, 1$
D	$1, 4$	$4, 3$

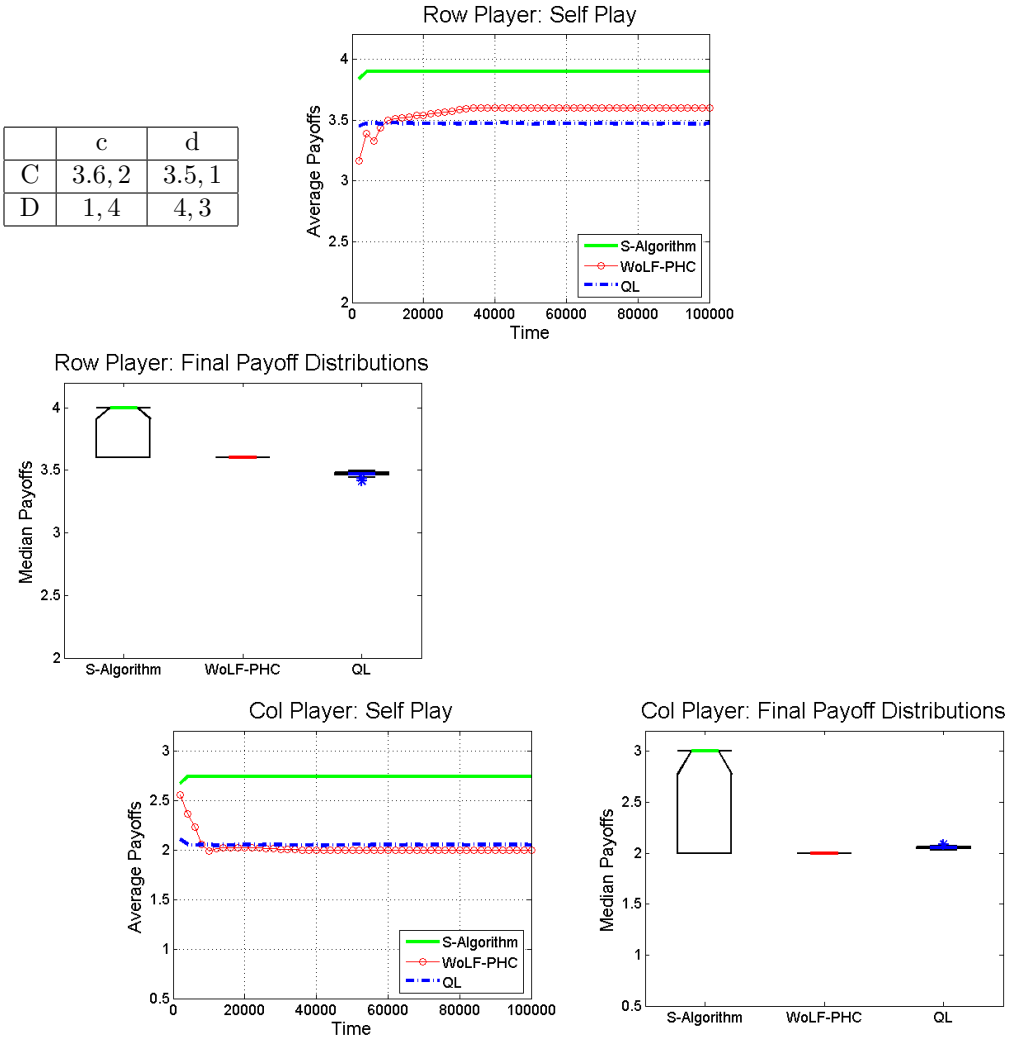


Figure 3.25: Results from 50 trials of the game shown at left. The performance of the row player is shown above and the performance of the column player is shown below.

The solutions the S-Algorithm learned in the large empirical study we presented in Section 3.5 where relatively *fair* in the sense that the agents tended to split the high payoffs (when possible). This result is closely tied to initial aspirations. Additionally, the S-Algorithm tends to learn the (fair) solutions suggested by Littman [53] when these solutions are pure strategies. When initial aspiration levels are similar between the agents, we would expect learned strategies to distribute payoffs more evenly among the agents then if they varied widely. Other factors also affect fairness, including learning rates and fallback positions.[18] Future work could include in-depth analysis of how these issues affect fairness and convergence to ε-pareto efficiency.

The S-Algorithm's fallback strategy (playing randomly when not satisfied) seems to be a key to the algorithm (which we have not discussed widely in this chapter), and deserves more study in future work. In games such as the iterated prisoner's dilemma, this fallback strategy induces other agents to cooperate since it essentially becomes a stochastic tit-for-tat [2]. However, in other games, this fallback strategy is not as effective. In addition to providing a method for issuing threats to defecting agents [54], a fallback strategy is essential to coordinating cooperation in games of conflicting interest. The S-Algorithm's fallback strategy is effective since it searches the reward space thoroughly. Future work should study alternate fallback strategies.

In this chapter, we focused on non-competitive general-sum games. This is largely because the focus of the chapter was pareto efficiency. In competitive games, all solutions are pareto efficient, and, thus, pareto efficiency is not a good metric of success. The S-Algorithm is not a good choice for an algorithm in competitive games. This is largely because the minimax solution (an agents security strategy) is a mixed strategy in many of these games, so the S-Algorithm cannot learn to play it. However, if the minimax strategy is a pure strategy, then the S-Algorithm can learn (and does [71]) to play effectively (somewhat) in these games. In Chapter 5, we discuss how an algorithm related to the S-Algorithm can incorporate security when the minimax strategy is known.

We also note that, in this chapter, we assumed a deterministic payoff function. While many multiagent learning algorithms (for example, WoLF-PHC and Q-learning) are designed to handle the situation in which payoffs are nondeterministic, the S-Algorithm is not. Again, in Chapter 5, we discuss an algorithm related to the S-Algorithm that can learn effectively when payoffs are nondeterministic.

[18] An agent's *fallback position* is the value that it guarantees itself when it plays its fallback strategy. A *fallback strategy* is the strategy that an agent plays when it is not satisfied.

Chapter 4

Satisficing Learning with Aspiration Trembles

4.1 Introduction

In the previous chapter we discussed the S-Algorithm (as presented by Stimpson *et al.* [70, 71]) in general-sum matrix games. We showed that under certain conditions (i.e., if the joint aspirations of the agents are guaranteed to remain in certain subregions for a positive number of episodes) the S-Algorithm learns with high probability to play ε-PE solutions[1]. Furthermore, we showed that in most general-sum matrix games (and all 2-agent matrix games) there exists a non-empty set of initial joint aspiration vectors such that these conditions are met.

Unfortunately, when the payoffs and actions of other agents in an arbitrary matrix game are unknown, the set of initial joint aspirations that satisfy the necessary conditions (for convergence to ε-PE with high probability) are also unknown. In most general-sum matrix games, this is not overly problematic since most *high* initial aspirations satisfy the conditions. However, in a small set of general-sum matrix games, these conditions are not easily met.

To overcome this difficulty, we extend the S-Algorithm to include periodic perturbations in aspiration levels, which we will call *aspiration trembles*. We call the new algorithm SAwT (*S-A*lgorithm *w*ith *T*rembles). This extension is in line with Karandikar's original aspiration-based satisficing learning algorithm [46]. In this chapter, we analyze aspiration trembles in repeated general-sum matrix games. In particular, we show that an agent can learn to tremble its aspirations to improve its payoffs. We show (empirically) that this learning process results in aspiration trembles that satisfy the conditions of pareto efficiency (if it is beneficial for the agents) in many games.

In addition to studying how aspiration trembles affect the performance of SAwT in stationary general-sum matrix games, we also consider the case in which payoff matrices vary over time. Situations in which payoff matrices change over time include learning in stochastic games, which we will discuss in Chapter 6. We sketch the argument that if payoff matrices change slowly enough, then SAwT will play ε-PE solutions high percentages of the time.

[1]Recall (see Section 3.4.3) that we refer to the conditions that lead to convergence to ε-PE (with high probability) as the *conditions of pareto efficiency*.

As in the previous chapter, we assume that an agent does not know the payoffs and actions of the other agents in the game. Thus, an agent does not know its own payoff matrix. Additionally, unless stated otherwise, we assume that an agent associates with a copy of itself (i.e., self play).

The structure of this chapter is as follows. In the next section (Section 4.2), we relate aspiration trembles to traditional exploration methods. We present an overview of SAwT in Section 4.3, and analyze three naive implementations (i.e., methodologies for trembling aspirations) of this algorithm. We show that, while these aspiration trembles do remove the requirement (of the S-Algorithm) that initial aspirations should be set *high*, they do not always satisfy the the conditions of pareto efficiency (although they do in most games). In Section 4.4, we show how a simple learning algorithm (such as Q-learning) can be used to learn how an agent should tremble its aspirations to improve its payoffs. In Section 4.5, we sketch the argument that appropriate implementations of SAwT continue to play ε-PE solutions even when the payoff matrix changes over time (as long as it does not change too quickly). Finally, we summarize and discuss these results in the last section.

4.2 Relating Aspiration Trembles to Traditional Exploration Methods

A main issue of standard reinforcement learning is the trade-off between exploration and exploitation [44]. While an agent must exploit the information that it has learned, it must also continue to explore its environment to guarantee that it learns to play *optimally*. In multiagent learning situations, this trade-off between exploration and exploitation becomes even more delicate [28].

Most traditional reinforcement learning algorithms require that an agent never cease exploring its environment. For example, Q-estimates in Q-learning [76] converge to their true Q-values (in stationary environments) only if each state-action pair is visited infinitely often. Thus, many implementations of the Q-learning algorithm force the agent to act randomly with a small probability for the duration of the agent's existence.

However, despite the importance of exploration methods in reinforcement learning, simple probabilistic exploration remains the state-of-the art [75]. Most of these methods fit into the category of ϵ-greedy methods, in which an agent explores with some probability, and exploits otherwise. Various implementations of ϵ-greedy exploration exist, including Boltzmann exploration, etc.

In contrast to traditional learning methods, the S-Algorithm turns exploration *on* and *off* depending on whether the agent is satisfied or not. Thus, in self play, once all agents are satisfied (i.e., at a satisficing equilibrium [74]), no more exploration takes place. It is because the S-Algorithm ceases to explore that it can only guarantee success with high probability. On the flip side, one can argue that the S-Algorithm is successful at learning ε-PE solutions (in self play) *because* it stops exploring.

The S-Algorithm can be extended to explore its environment in two ways. The first way is to allow the agent to periodically act randomly (i.e., an ϵ-greedy-like policy). We call this method *action exploration*. Action exploration induces a small perturbation in an agent's aspirations[2] when the random action yields a different payoff than the agent has been receiving (when it is satisfied). These small trembles in aspirations can cause additional exploration as changes in aspirations can result in the agent being satisfied with different outcomes.

The second way in which the S-Algorithm can be extended to more thoroughly explore its environment is to use aspiration trembles. That is, an agent periodically trembles its aspirations according to some distribution $g(\alpha_i^t)$. This is the method used by Karandikar *et al.* [46]. When the distribution $g(\alpha_i^t)$ is tightly centered around α_i^t, then this method is essentially identical to action exploration. However, action exploration is a subset of aspiration trembles (in the case of the S-Algorithm) since $g(\alpha_i^t)$ can also be designed to cause large changes in aspirations.

Both small and large aspiration trembles appear to offer different advantages and disadvantages. In the next section, we discuss these differences and present several case studies showing how small and large aspiration trembles affect the performance of aspiration-based satisficing learning agents.

4.3 The S-Algorithm with Trembles

In this section, we present a basic overview of the S-Algorithm with Trembles (SAwT). Additionally, we present several naive implementations of SAwT, and show via several case studies that they eliminate the need to initialize aspirations to high levels. However, while these naive implementations of SAwT do learn to play ε-PE solutions in most games, these naive implementations of aspiration trembles are not guaranteed to satisfy the conditions of pareto efficiency (discussed in the previous chapter).

4.3.1 Basic Overview

SAwT, as is outlined in Algorithm 4.1, is identical to the S-Algorithm (see Algorithm 3.2) except that a) it does not restrict initial aspirations and b) it trembles its aspirations with probability η (according to the distribution $g(\alpha_i^t)$) if certain constraints are met. These constraints can vary from implementation to implementation. We offer some reasonable constraints in Section 4.3.1.

Observe that SAwT still differs from Karandikar *et al.*'s original aspiration-based satisficing algorithm in two ways. First, SAwT (like the S-Algorithm) does not incorporate inertia. Second, Karandikar allows the agents to tremble their aspirations

[2]The magnitude of aspiration perturbations (per episode) using action exploration is bounded by λW, where W is the maximum difference between aspirations and payoffs, given by $W = \max(r_i^{\max} - \alpha_i^t, \alpha_i^t - r_i^{\min})$, where $r_i^{\max}$ and $r_i^{\min}$ are the maximum and minimum payoffs in agent i's payoff matrix.

Repeat

(a) Select an action a_i^t according to the following criteria:

$$a_i^t \leftarrow \begin{cases} a_i^{t-1} & \text{if } (\alpha_i^{t-1} \leq r_i^{t-1}) \\ \text{rand}(A_i) & \text{otherwise} \end{cases}$$

where $\text{rand}(A_i)$ denotes a random selection from the set A_i.

(b) Receive reward r_i^t and update aspiration level:

if *constraints met*, then w/ prob η

$\alpha_i^{t+1} = g(\alpha_i^t)$

else

$\alpha_i^{t+1} \leftarrow \lambda\alpha_i^t + (1-\lambda)r_i^t$

Algorithm 4.1: The S-Algorithm with Trembles (SAwT) for agent i.

at any time, whereas SAwT places constraints on when an agent can tremble. These differences are necessary if we are to apply this form of learning to general-sum matrix games (rather than just the small class of games Karandikar *et al.* considered).

Below we discuss various methods for deciding *when* (i.e., what are the constraints) and *how* (i.e., what form should $g(\alpha_i^t)$ take) an agent should tremble its aspirations.

Constraints on Aspiration Trembles

An infinitely repeated matrix game played by SAwT agents is an infinite sequence of *epochs*. An epoch is a set of consecutive episodes beginning and ending with an aspiration tremble by any of the agents in the game. The behavior of the agents in each episode can be divided into two phases, or time periods. We refer to the first time period as the *relaxation search*. During this phase, each agent searches for a sustainable satisficing outcome as it relaxes its aspirations toward its average payoffs[3]. The relaxation search commences when one of the agents trembles its aspirations and concludes (for each agent) when the agent's strategy converges[4]. The *convergence period* follows the relaxation search and ends (as does the epoch) when one of the agents once again trembles its aspirations. Note that it is possible for an epoch to not have a convergence period if the agents do not converge before one of the agents trembles its aspirations. However, this change in epochs may not be discernible to the agents that do not tremble.

[3]Agent i's average payoffs during the relaxation search can be approximated by $\frac{1}{|A|}\sum_{\mathbf{a}\in A} r_i(\mathbf{a})$ (though this estimate is not completely accurate).

[4]Note that we have altered the definition of *convergence* from the previous chapter. In this chapter, we say that an agent has converged if its strategy will remain unchanged until the end of the current epoch.

Since the primary purpose of aspiration trembles is to find a better solution than the one currently being played, we contend that an agent should not tremble its aspirations during the relaxation search unless it has cause to believe that the relaxation search will not end, or that it will not end satisfactorily. Thus, all aspiration trembles should occur during the convergence period of an epoch unless, again, an agent has reason to believe that the epoch will not have a convergence period or, if it does, the convergence period will not be profitable.

This means that a SAwT agent must determine two things. First, an agent must be able to determine when the relaxation phase of an epoch has concluded (i.e., play has converged). Second, prior to the convergence of the relaxation search, it must identify whether or not play is likely to converge to a profitable (for it) solution.

SAwT can conclude with some (hopefully high) probability that the relaxation search concluded at time T if two conditions have held for all $t \in [T, T + T^c)$, where T^c is a positive constant. The first condition is that the agent has been satisfied for all $t \in [T, T + T^c)$. The second condition is that $r_i^t = r_i^T$ for all $t \in [T, T + T^c)$[5]. We note that T^c controls how certain an agent is that play has actually converged. If T^c is too small, then an agent may conclude prematurely that play has converged.

To summarize, SAwT concludes that the relaxation search for a given epoch has concluded at time T if

$$\forall t \in [T, T + T^c), r_i^t \geq \alpha_i^t \text{ and } r_i^t = r_i^T \tag{4.1}$$

We now turn to the issue of determining whether or not play is likely to converge to a profitable solution (in a given epoch). Relaxation searches that result in profitable payoffs during the convergence period are usually shorter than those that do not. This is because aspirations tend to move toward average payoffs during the relaxation search. Thus, the longer the relaxation search, the more likely (in general) an agent is to be content with average payoffs. Additionally, a long relaxation search may be the result of a preclusion critical event that prohibits convergence (to profitable solutions). For these reasons, an agent should consider trembling if the relaxation search has not concluded in a *reasonable* number of episodes (which we denote $T^r(\lambda)$). Note that $T^r(\lambda)$ is dependent on λ since aspirations change more slowly for high values of λ than for smaller values.

We now discuss how to determine $T^r(\lambda)$. Let T_0 be the time at which the agent believes the current epoch began. Assuming that play has not converged, agent i's aspiration level at time t is (from the aspiration updated equation) approximately

$$\alpha_i^t \approx \lambda^{t-T_0} \alpha_i^{T_0} + (1 - \lambda^{t-T_0}) \bar{r}_i \tag{4.2}$$

[5]In the event of an unstable security critical event, a SAwT agent's strategy may have converged even though an action continues to yield different (yet satisficing) payoffs. However, we do not concern ourselves with this situation since making this assumption does not hinder the performance of the agent. Note also that this observation assumes a deterministic payoff function (which the S-Algorithm also assumes).

where $\bar{r}_i$ is agent i's average payoff. Thus, λ^{t-T_0} approximates the distance the current aspirations are between $\alpha_i^{T_0}$ and $\bar{r}_i$. Thus, the longer the relaxation search lasts (i.e., the bigger $t - T_0$ gets), the closer agent i's average aspirations will be to its average payoff $\bar{r}_i$ and the less its aspirations will be like $\alpha_i^{T_0}$.

Let $\theta \in (0, 1)$ be a threshold on how far (i.e., percentage) that an agent will allow its aspirations to change before it determines that the relaxation search is not likely to end profitably. Then, the number of episodes that a relaxation search should last before an agent considers trembling is given by solving for $T^r(\lambda)$ in the equation $\lambda^{T^r(\lambda)} = \theta$, which yields

$$T^r(\lambda) = \frac{\log \theta}{\log \lambda} \tag{4.3}$$

Thus, when $t - T_0 > T^r(\lambda)$, an agent should consider trembling its aspirations. Note that the closer θ is to 0, the more likely the relaxation search will not converge to a profitable solution (if at all), since aspirations will get closer to average payoffs.

When probabilistic methods for determining aspiration trembles are used, typically only one agent will tremble at a given time. However, in most cases, this is all that is necessary to trigger a relaxation search, as all agents can become dissatisfied when just one agent changes its aspiration level (assuming the agents are satisfied prior to the tremble). If aspirations are trembled to a large degree (in the upward direction), then the trembling agent (i.e., the agent that trembles) will have an advantage in many games over the other agents since it will be more selective (i.e., satisfied with less outcomes) than the other agents. Knowing this, all agents have an incentive (in many games) to tremble their aspirations higher and more frequently than necessary,[6] which will result in lower average payoffs to all the agents.

One way of circumventing this problem is for all agents to tremble their aspirations at the same time (approximately). SAwT agents can do this without explicitly communicating if trembles occur during the convergence period. If an aspiration tremble occurs in the convergence period, then all the other agents' payoffs will usually change suddenly, at which point each agent should tremble its aspirations.

We call such aspiration trembles *coordinated trembles*. Coordinated trembles can be viewed as trigger strategies[7] which can be used to create profitable rNEs, and are the basis for the proof of the folk theorem [35].

The Tremble Distribution $g(\alpha_i^t)$

Karandikar *et al.* [46] allowed aspirations to be trembled to any real number in a compact interval Λ, where Λ contained all *feasible payoffs*. This keeps aspirations

[6] We note that agents can obtain similar effects by lowering their learning rates (i.e., increasing λ).

[7] A trigger strategy is a strategy that switches suddenly (usually with the intent to punish another agent) when another agent does not play as expected.

from wandering too far from the payoffs of the game. While maintaining this restriction, we desire to understand how several issues related to aspiration trembles affect the performance of SAwT. These issues include coordinated verses non-coordinated trembles as well as the magnitude of trembles (i.e., large trembles verses small trembles).

Both small and large trembles appear to offer different advantages. Specifically, small trembles will generally result in shorter relaxation searches. This seems to be desirable since the agents' average payoffs received during the relaxation search are usually lower than those received during the convergence period. On the flip side, large trembles in the upward direction usually give an agent a better chance of receiving a high payoff when strategies converge.

While Karandikar *et al.* allowed aspiration trembles in either direction (up or down), we consider only trembles in the upward direction in this section. This restriction seems reasonable if agents only tremble when satisfied, since downward trembles will have no affect on the immediate behavior of the agents [37]. However, we note that downward trembles may be desirable when agents coordinate trembles. We consider the possibility of downward trembles in the next section.

4.3.2 Case Studies

Based on the previous discussions, we designed three naive implementations of SAwT. We call these implementations SAwST (*S*-*A*lgorithm with *S*mall *T*rembles), SAwLT (*S*-*A*lgorithm *w*ith *L*arge *T*rembles), and SAwCT (*S*-*A*lgorithm *w*ith *C*oordinated *T*rembles). After presenting the details of the implementations, we will compare them in several matrix games.

Since the algorithms differ only in their specifications of step (b) of Algorithm 4.1, we only present each algorithm's specific implementation of this step.

SAwST. SAwST is given in Algorithm 4.2. In SAwST, an agent perturbs its current aspirations with probability η if it a) determines that the relaxation search has completed (the first condition of the if statement) or if it b) believes that the relaxation search will not result in convergence to a profitable solution (the second condition of the if statement). The perturbation ξ is a real number randomly selected from the interval $[0, 0.00001]$.

SAwLT. SAwLT is shown in Algorithm 4.3. In SAwLT, an agent trembles its aspirations under the same conditions as in SAwST. The difference is that SAwLT trembles its aspiration level to the highest payoff it has seen up to that point (denoted by $r_i^{\max}$) plus, a small positive margin ϵ.

SAwCT. SAwCT, shown in Algorithm 4.4, is identical to SAwLT except that it also trembles (with probability 1) when it believes that another agent has trembled (i.e., it uses coordinated trembles). SAwCT determines that another agent has trembled its aspirations if $r_i^t \neq r_i^{t-1}$ and $r_i^{t-1} = r_i^j$, for all $j \in [t - T^c, t - 1)$ (i.e., the *else*

(b) Receive reward r_i^t and update aspiration level:
if $\left(\forall \upsilon \in (t - T^c, t], r_i^\upsilon \geq \alpha_i^\upsilon \text{ and } r_i^\upsilon = r_i^t\right)$ or $(t - T_0 > T^r(\lambda))$ then w/ prob η
$\alpha_i^{t+1} = \alpha_i^t + \xi$
else
$\alpha_i^{t+1} \leftarrow \lambda \alpha_i^t + (1 - \lambda) r_i^t$

Algorithm 4.2: Aspiration update rule for the S-Algorithm with Small Trembles (SAwST) for agent i. ξ is randomly selected from the interval $[0, 0.00001]$. T^c is a constant defined in the arguments surrounding Equation (4.1), T_0 is the time at which an agent believes the current epoch began, and $T^r(\lambda)$ is given by Equation (4.3).

(b) Receive reward r_i^t and update aspiration level:
if $\left(\forall \upsilon \in (t - T^c, t], r_i^\upsilon \geq \alpha_i^\upsilon \text{ and } r_i^\upsilon = r_i^t\right)$ or $(t - T_0 > T^r(\lambda))$ then w/ prob η
$\alpha_i^{t+1} = r_i^{\max} + \epsilon$
else
$\alpha_i^{t+1} \leftarrow \lambda \alpha_i^t + (1 - \lambda) r_i^t$

Algorithm 4.3: Aspiration update rule for the S-Algorithm with Large Trembles (SAwLT) for agent i. ϵ is a small positive constant. T^c is a constant defined in the arguments surrounding Equation (4.1), T_0 is the time at which an agent believes the current epoch began, and $T^r(\lambda)$ is given by Equation (4.3).

if statement of Algorithm 4.4).

For each implementation, the tremble rate η was initially set to 0.01 and then decreased slowly over time using

$$\eta = 0.01^{\left(\frac{\tau+200}{200}\right)} \tag{4.4}$$

where τ is the current epoch (estimated by the number of times the agent has trembled its own aspirations).

Each agent's initial aspiration level was randomly selected from the set of reals in the interval $[5, -5]$. Also, unless specified otherwise, $\lambda = 0.99$, $\theta = 0.1$, and $T^c = 20$.

We compare these three implementations in five matrix games. These games are chicken, staghunt, MASD ($k = 0.6$, $n = 3$, $M = 5$), and two other matrix games that have stable security and preclusion critical events. For each game we show (as in the previous chapter) the average payoffs of the agents over time (average of 50 trials) as well as a box and whiskers plot of the average payoffs of the agents in the last 0.1% of the episodes.

(b) Receive reward r_i^t and update aspiration level:
if $\left(\forall \upsilon \in (t - T^c, t], r_i^\upsilon \geq \alpha_i^\upsilon \text{ and } r_i^\upsilon = r_i^t\right)$ or $(t - T_0 > T^r(\lambda))$ then w/ prob η
$\alpha_i^{t+1} = r_i^{\max} + \epsilon$
else if $r_i^t \neq r_i^{t-1}$ and $r_i^{t-1} = r_i^j$, $\forall j \in [t - T^c, t - 1)$
$\alpha_i^{t+1} = r_i^{\max} + \epsilon$
else
$\alpha_i^{t+1} \leftarrow \lambda \alpha_i^t + (1 - \lambda) r_i^t$

Algorithm 4.4: Aspiration update rule for the S-Algorithm with Coordinated Trembles (SAwCT) for agent i. T^c is a constant defined in the arguments surrounding Equation (4.1), T_0 is the time at which an agent believes the current epoch began, and $T^r(\lambda)$ is given by Equation (4.3).

Chicken

Results for the game chicken are shown in Figure 4.1. Each of the three SAwT implementations learn to play the PE solution (C, c) in this game, yielding a payoff of 3 to each agent. Notice that the average reward also converges to 3 despite the cost incurred by relaxation searches, even for large aspiration trembles. This is because the probability of aspiration trembles (η) decreases over time.

Staghunt

As in the matrix game chicken, the three SAwT agents also learn to play cooperatively in staghunt (see Figure 4.2). However, it takes much longer for SAwST (small trembles) to converge to mutual cooperation. This is because small aspiration trembles make it less probable for both agents to become dissatisfied with mutual defection (i.e., (D, d)) at the same time (which is required for the agents to overcome the security critical event of playing D and d). Thus, it takes longer for them to learn to play the more profitable solution (C, c).

MASD

The performance of the agents in a 3×6 MASD ($k = 0.6$) is shown in Figure 4.3. Results are shown for a slower learning rate ($\lambda = 0.999$). In this game, small aspiration trembles (SAwST) are just as effective as large coordinated trembles (SAwCT). However, large non-coordinated trembles result in lower average performance. This is not because SAwLT does not learn to play PE solutions. Rather, it is a result of the agent learning PE solutions with lower average payoffs, as one of the agents (the trembling agent) typically receives a higher payoff than the non-trembling agents.

	c	d
C	$3, 3$	$2, 3.5$
D	$3.5, 2$	$1, 1$

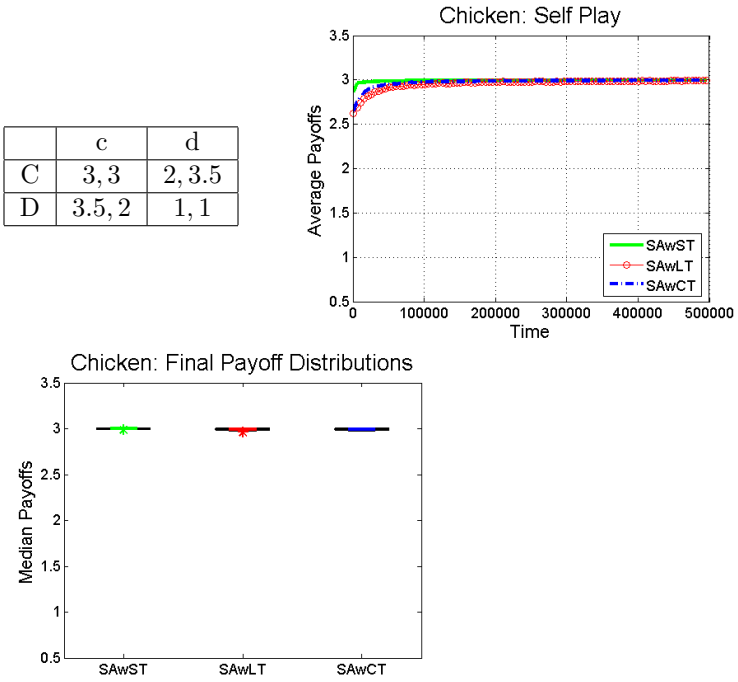


Figure 4.1: Results from 50 trials of the matrix game chicken.

	c	d
C	$4, 4$	$-5, 3$
D	$3, -5$	$2, 2$

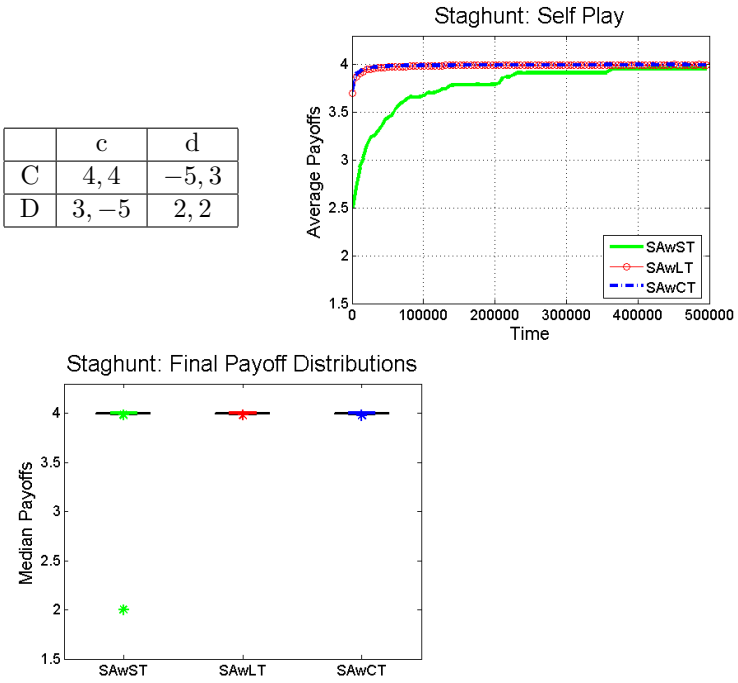


Figure 4.2: Results from 50 trials of the matrix game staghunt.

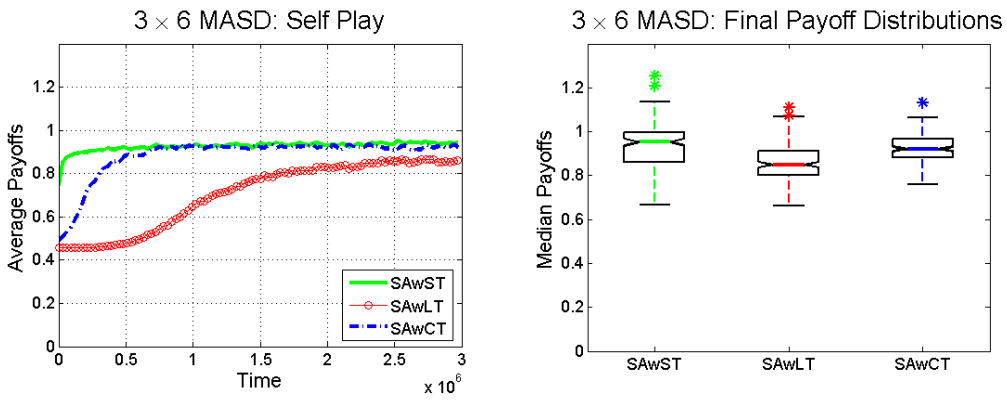


Figure 4.3: Results from 50 trials of a 3-agent, 6-action MASD (k=0.6). The results are shown for $\lambda = 0.999$.

This result is in line with Stimpson *et al.*'s observation that aspirations should be similar in the MASD.

A Game with a Stable Security Critical Event

Figure 4.4 (left) shows a matrix game with a stable security critical event (when player 1 plays A) when player 1's aspiration level is less than or equal to 3.5. The S-Algorithm (as we saw in the previous chapter; see Figure 3.24) converges to the pareto dominated solution (A, a) in this game for most initial aspiration vectors. However, if $\alpha^0 \approx (4, 3)$, then the S-Algorithm converges with high probability to the PE solution (B, b), resulting in a higher payoff to both agents.

Figure 4.4 (right) shows that naive implementations of SAwT still converge to the pareto dominated solution (A, a) much of the time, particularly in the case of SAwST (small trembles). However, large aspiration trembles do increase the average performance of the agents (slightly). Figure 4.4 (right) also shows the performance of SAwT-Fixed, which is identical to SAwCT except that the agents always tremble their aspirations to the point $(3.99, 2.99)$. Such trembles result in convergence to the PE solution (B, b).

A Game with a Preclusion Critical Event

We now analyze SAwT in a game with a preclusion critical event. A similar version of this game was discussed in the previous chapter (see Figure 3.12). Once again, SAwST is outperformed by the algorithms that use larger trembles. However, there appears to be little difference between SAwCT and SAwLT. This is because any aspiration tremble to a region in which one of the agents has a payoff greater than 3 satisfies the conditions of pareto efficiency.

	a	b
A	$3.5, 2$	$3.5, 1$
B	$1, 4$	$4, 3$

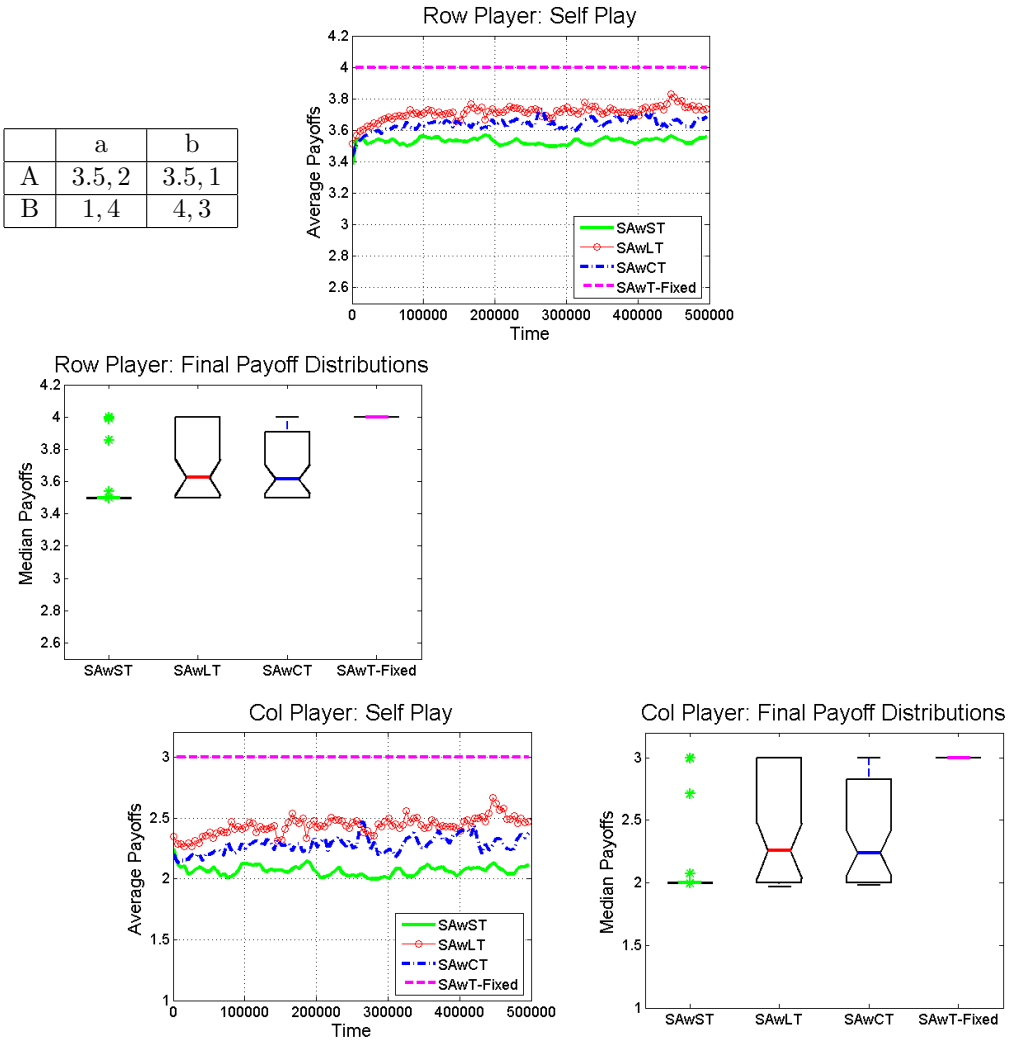


Figure 4.4: Results from 50 trials of the game shown at left. The performance of the row player is shown above and the performance of the column player is shown below. SAwT-Fixed is a SAwT agent that always trembles (using coordinated trembles) to the point $(3.99, 2.99)$.

	a	b	c
A	$0, 3$	$3, 0.5$	$3, 0$
B	$0.5, 3$	$3.3, 3.3$	$1.5, 3$
C	$3, 1.5$	$3, 1$	$1, 3$

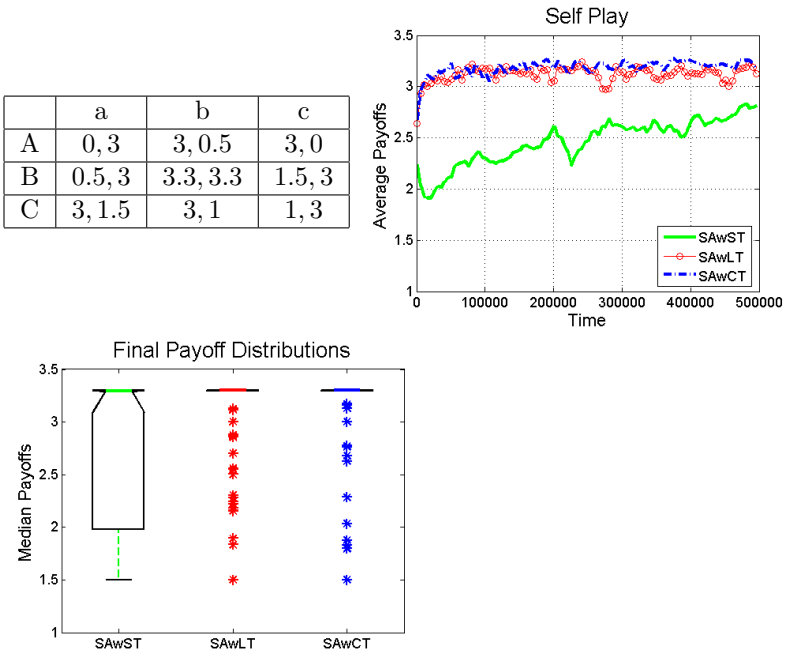


Figure 4.5: Results from 50 trials of the game shown at left.

4.3.3 Discussion

In each of the games discussed in this section, each implementation of SAwT performs (after learning) as well or better than the S-Algorithm (with the exception of SAwST in the game shown in Figure 4.5). Thus, aspiration trembles (particularly large ones) eliminate the need for the satisficing learners to set their initial aspiration levels to high values.

In several of the games we analyzed in this section, large aspiration trembles tend to lead to better overall performance than do small aspiration trembles (see Figures 4.4 and 4.5). There are several reasons for this. First, in many games, large aspiration trembles allow the agents to tremble their aspirations to locations in the payoff space that satisfy the conditions of pareto efficiency discussed in the previous chapter. Small aspiration trembles do not do as good of a job of overcoming critical events that exclude mutually satisficing ε-PE solutions. Second, although smaller aspiration trembles usually result in shorter relaxation searches, this advantage is negated as the probability of aspiration trembles becomes infrequent (i.e., as $\eta \to 0$).

In most games we studied in this section, coordinating trembles does not increase the average payoffs of the agents. However, in the case of large aspiration trembles, coordinating trembles does have a significant impact on the performance of the agents in the MASD. This is because non-trembling agents eventually become satisfied with lower payoffs when one of the agents becomes more selective (due to a large aspiration

tremble). This results in lower average payoffs to the agents.

Unfortunately, naive aspiration trembles (at least those we presented) do not satisfy the conditions of pareto efficiency in all games (see, for example, Figure 4.4). As a result, the agents receive lower average payoffs (at least in these games) than they otherwise might. In the next section, we show how agents can use a simple learning algorithm to learn how to tremble their aspirations so as to improve their average payoffs. We show (empirically) that this algorithm allows agents to learn to tremble aspirations so that they satisfy the conditions of pareto efficiency (when doing so is profitable).

4.4 Using Trembles to Learn Initial Aspirations

In the previous section, we considered trembling aspirations to the highest observed payoff or perturbing aspirations slightly in the upward direction. In this section, we consider trembling aspirations to a set of fixed locations in the payoff space. Trembling aspirations in this way allows an agent to learn how specific initial aspiration levels (of a given epoch) affect future payoffs.

4.4.1 Background and Notation

At the beginning of an epoch, each agent trembles its aspiration level to one of $Z_i + 1$ values, where Z_i is some positive integer. Let $\mathcal{C}_i$ denote agent i's set of *tremble values* (aspiration levels to which agent i considers trembling its aspirations). Agent i's maximum tremble value (denoted $c_i^{Z_i} \in \mathcal{C}_i$) is $c_i^{\max} = r_i^{\max} + \frac{\varepsilon}{2}$ where $r_i^{\max}$ is agent i's highest observed payoff. Its minimum tremble value (denoted $c_i^0 \in \mathcal{C}_i$ is $c_i^{\min} = m_i^P - \frac{\varepsilon}{2}$, where m_i^P (as in the previous chapter; see Equation (3.7)) is its minimax value (over the pure strategy space) given by

$$m_i^P = \max_{a_i \in A_i} \min_{\mathbf{a}_{-i} \in A_{-i}} r_i(a_i, \mathbf{a}_{-i}) \tag{4.5}$$

Note that m_i^P is unknown to agent i since it does not know its payoff matrix. However, agent i can calculate an upper bound of m_i^P (which must converge to the true value over time if all the agents sometimes act randomly) using observations of the minimum payoff produced by each of its actions.

Let Δ_i be the distance between each tremble value, given by

$$\Delta_i = \frac{c_i^{\max} - c_i^{\min}}{Z_i} = \frac{r_i^{\max} - m_i^P + \varepsilon}{Z} \tag{4.6}$$

Then, each tremble value $c_i^k \in \mathcal{C}_i$ of agent i is given be

$$c_i^k = m_i^P - \frac{\varepsilon}{2} + k\Delta_i \tag{4.7}$$

We consider the case in which agents coordinate aspiration trembles (as in SAwCT; see Algorithm 4.4). Using this method, the initial joint aspirations of each epoch are

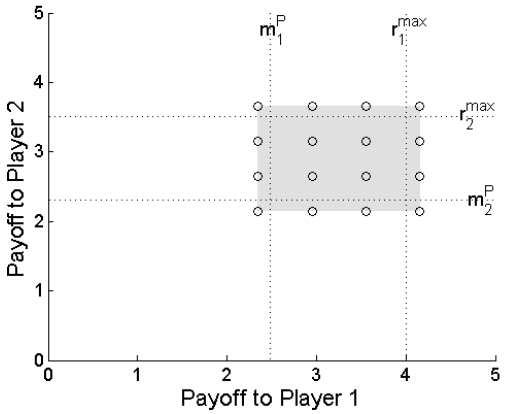


Figure 4.6: Joint tremble values (marked with o's) for an arbitrary 2-agent matrix game for $Z = 3$, $\varepsilon = 0.3$, $m_1^P = 2.5$, $m_2^P = 2.3$, $r_1^{\max} = 3.5$, $r_2^{\max} = 4$.

in the region $\mathbb{P}_0$ (the region of possible initial aspiration vectors defined in the previous chapter). Let $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$ be the set of *joint tremble values*[8]. Figure 4.6 shows the set of joint tremble values for an arbitrary 2-agent matrix game. The o's in the figure mark the joint tremble values of the agents when $Z_i = 3$ (for both agents).

A SAwT agent should learn to tremble its aspirations to maximize its own convergence payoffs. In most games, joint tremble values that satisfy the conditions of pareto efficiency (if the agents tremble their aspirations to this point) are most likely to satisfy this requirement. Thus, the question arises of how large Z_i should be (for each agent i) to guarantee that at least one joint tremble value will satisfy the conditions of pareto efficiency. If one such joint tremble value exists, then the agents can learn to tremble their aspirations to this point in order to maximize their convergence payoffs.

In most games, Z_i need not be large as most joint tremble values in $\mathbb{P}_0$ satisfy the conditions of pareto efficiency. In others, Z_i must be chosen more judiciously. In others still, it appears that there is no possible joint tremble value that satisfies the conditions of pareto efficiency (see the previous chapter). We now present a claim that places a bound on how high Z_i must be to guarantee that at least one joint tremble value will satisfy the conditions of pareto efficiency in most games.

Claim 4.1. *If $Z_i \geq 1 + \frac{c_i^{\max} - c_i^{\min}}{\varepsilon}$ for each agent i, then at least one $\mathbf{c} \in \mathcal{C}$ satisfies the conditions of pareto efficiency if the game being played has one of the following characteristics:*

(a) The game has no preclusion critical events.

(b) The game is a 2-agent game.

[8]Note that $\mathcal{C}$ is only approximately the set of initial joint aspirations of each epoch since the agents do not all tremble on the same episode when trembles are coordinated.

(c) $\exists \mathbf{a} \in \mathbb{P}_\varepsilon$ *such that* $\forall i\ r_i(\mathbf{a}) > m_i^P$ *and* $\forall J \subseteq I\ (|J| \geq 2),\ \exists j \in J\ r_j(\mathbf{a}) > T_j^{\text{prec}} + \varepsilon$.

In each of these cases (Cases (a)–(c)), if $Z_i = 1 + \frac{c_i^Z - c_i^0}{\varepsilon}$ for each agent i, then a joint tremble value must be in a subregion defined in the proofs of the theorems of the previous chapter. The subregion for Case (a) is given in the proof of Theorem 3.2, the subregion for Case (b) is given in the proof of Theorem 3.3, and the subregion for Case (c) is given in the proof of Theorem 3.4.

4.4.2 Learning Tremble Actions

Each agent i seeks to learn which of its tremble values gives it the highest average payoff once play converges (if it does, indeed, converge). We call this payoff the *convergence payoff*. Formally, let $c_i(\tau)$ denote agent i's selected tremble value in the τ^{th} epoch and let $\mathbf{c}(\tau) = (c_1(\tau), \cdots, c_n(\tau))$ be the joint tremble value (of the n agents) in the τ^{th} epoch. The joint tremble value $\mathbf{c}(\tau)$ determines the convergence payoffs that the agents will likely receive in epoch τ. We denote this payoff u_i^τ for agent i. For practical purposes, let $u_i^\tau = \alpha_i^{T_{\text{end}}^\tau}$, where T_{end}^τ is the last episode of epoch τ.

The convergence payoff u_i^τ is a sample taken from a distribution over the real numbers. Let this distribution be denoted $\mathcal{U}_i(\mathbf{c}(\tau))$. Then, the tuple $(I, \mathcal{C}, \mathcal{U})$ (where $\mathcal{U} = \{\mathcal{U}_1, \cdots, \mathcal{U}_n\}$ and I, as before, is the set of n agents playing the game) is a matrix game, which we will call the *tremble matrix game*.

In contrast to the matrix games we have studied so far, tremble matrix games have nondeterministic payoff functions. Because of this, the S-Algorithm is not a good choice for learning this game. Thus, we must select an alternate learning algorithm. Fortunately, empirical evidence strongly suggests that the tremble matrix games does not take on the form of all general-sum matrix games, as (it appears) each agent has a tremble value (or set of tremble values) that dominates the other tremble values (i.e., regardless of how the other agents choose to tremble their aspirations). Additionally, it does not appear that off-dominant solutions can lead to higher payoffs than dominant solutions (such as cooperation in the prisoner's dilemma). Thus, a simple single-agent learning algorithm appears to be sufficient to learn in these games.

In this chapter, we explore the effectiveness of a stateless Q-learning [76] agent to learn the tremble matrix game. We now describe our implementation (which we call SAwT-Q).

Updating Q-values

SAwT-Q keeps track of two different Q-estimates for each tremble value $c_i \in \mathcal{C}_i$: $Q_i^\tau(c_i|\text{down})$ and $Q_i^\tau(c_i|\text{up})$. $Q_i^\tau(c_i|\text{down})$ is the estimate of the agent's average convergence payoff (at epoch τ) when it trembles its aspirations downward to the tremble value c_i, while $Q_i^\tau(c_i|\text{up})$ is the estimate of the agent's average convergence payoff when it trembles its aspirations upward to the tremble value c_i.

At the end of each epoch, one Q-update is made. Let T_0 be the time at which the current epoch began. Then, if $\alpha_i^{T_0} \geq c_i(\tau)$ (i.e., the agent trembled downward), then $Q_i^\tau(c_i|\text{down})$ is updated using

$$Q_i^{\tau+1}(c_i(\tau)|\text{down}) = (1-\rho)Q_i^\tau(c_i(\tau)|\text{down}) + \rho u_i^\tau \tag{4.8}$$

Otherwise, $Q_i^\tau(c_i|\text{up})$ is updated using

$$Q_i^{\tau+1}(c_i(\tau)|\text{up}) = (1-\rho)Q_i^\tau(c_i(\tau)|\text{up}) + \rho u_i^\tau \tag{4.9}$$

where ρ is the learning rate. For all results presented in this chapter, we used $\rho = \max(0.01, \frac{1}{\kappa(c_i(\tau))})$, where $\kappa(c_i(\tau))$ is the number of times that the agent has trembled its aspirations to the tremble value $c_i(\tau)$.[9]

Selecting Actions

We use an ϵ-greedy policy. That is, on epoch τ, the tremble value with the highest Q-estimate is selected with probability $1-\epsilon$. Otherwise (i.e., with probability ϵ), the tremble value is selected randomly. More formally,

$$c_i^\tau \leftarrow \begin{cases} \max_{c_i \in \mathcal{C}_i} Q_i^\tau(c_i) & \text{with probability } 1-\epsilon \\ \text{rand}(\mathcal{C}_i) & \text{with probability } \epsilon \end{cases} \tag{4.10}$$

where $\text{rand}(\mathcal{C}_i)$ is a random selection from the set $\mathcal{C}_i$ and $Q_i^\tau(c_i)$ is given by

$$Q_i^\tau(c_i) = \begin{cases} Q_i^\tau(c_i|\text{down}) & \text{if } \alpha_i^t \geq c_i \\ Q_i^\tau(c_i|\text{up}) & \text{if } \alpha_i^t < c_i \end{cases} \tag{4.11}$$

When the agent acts according to its Q-estimates, then η_i is decreased over time using

$$\eta_i = 0.01^{\left(1.0 + \frac{\tau}{200 Z_i}\right)} \tag{4.12}$$

where τ is the current epoch. However, when an agent explores its environment, lower tremble payoffs often result. To avoid receiving these payoffs for too long, we set $\eta = 0.01$ in these epochs (when the tremble value is chosen randomly). This way, SAwT gets the best of both worlds: it is able to explore its environment (as it is still able to observe its convergence payoff), but it is also able to exploit what it has learned for longer periods of time than when it explores.

As before, unless stated otherwise, $\lambda = 0.99$, $\theta = 0.1$ (for Equation (4.3)), and $T^c = 20$ (see Equation (4.1)). Also, $Z_i = 6$ for all i and $\epsilon = \eta_i$ (as given in Equation (4.12)).

The resulting algorithm (SAwT-Q) is a blend of the SAwT and Q-learning. SAwT (in its original unmodified form) is used to learn to play satisficing equilibrium [74] (or mutually satisficing solutions). Q-learning is used to learn how an agent can maximize the effect of trembling (by learning how tremble values affect the long-term performance of the agent).

[9] We distinguish between *up* and *down* trembles when computing $\kappa(c_i(\tau))$.

4.4.3 Results

We now present a case study showing (empirically) how SAwT-Q performs in a number of games. We compare the performance of SAwT-Q with two other agents in each game. First, we compare SAwT-Q's performance to SAwRT, an SAwT agent that randomly selects a tremble value from the set $\mathcal{C}_i$. Second, we compare SAwT-Q's performance to an agent that deterministically trembles its aspirations to locations that produce high convergence payoffs (SAwCT in most games). Both of these agents are designed to tremble their aspirations with the same frequency as SAwT-Q (i.e., they tremble their aspirations with probability η_i as given in Equation (4.12)).

Chicken

All initial aspiration vectors in $\mathbb{P}_0$ lead to convergence to an ε-PE solution in the matrix game chicken. However, we show the performance of SAwT-Q in this game to show that it learns to tremble its aspirations so that it converges to the mutually satisficing solution (C, c), which is the highest *average* payoff an agent can expect to receive in self play. Figure 4.7 shows that the performance of SAwT-Q is nearly identical to that of SAwCT. Notice, however, that SAwRT plays cooperatively (and does so very quickly) in this game (in most cases) despite the fact that it trembles its aspirations randomly. This is because most aspiration trembles to the tremble values in $\mathcal{C}_i$ tend to be in the downward direction. Since such trembles often have no affect on the behavior of SAwT agents (since the agents remain satisfied), SAwRT does not have as many aspiration relaxation searches (which can be costly). Thus, its average performance is higher than that of SAwT-Q and SAwCT until η becomes small.

MASD

The results for the 3×6 MASD (shown in Figure 4.8) are uneventful, yet interesting nonetheless. First, notice that the eventual performance of all the agents is nearly identical. However, SAwRT outperforms the other two agents initially. Again, this is due to aspiration trembles in the downward direction which do not result in behavior changes in the agents. Notice also that the performance of SAwT-Q is identical to that of SAwCT, since SAwT-Q agents learn to tremble using the tremble value $c_i^{\max}$ (in most instances).

A Game with a Stable Security Critical Event

In the previous two examples, learning how to tremble aspirations does not appear to increase the performance of the agents to a significant degree. However, SAwT-Q often outperforms naive implementations of SAwT in games in which the conditions of pareto efficiency are not easily satisfied. To see this, consider Figure 4.9, which shows a game with a stable security critical event. Recall from the previous section

	c	d
C	$3, 3$	$2, 3.5$
D	$3.5, 2$	$1, 1$

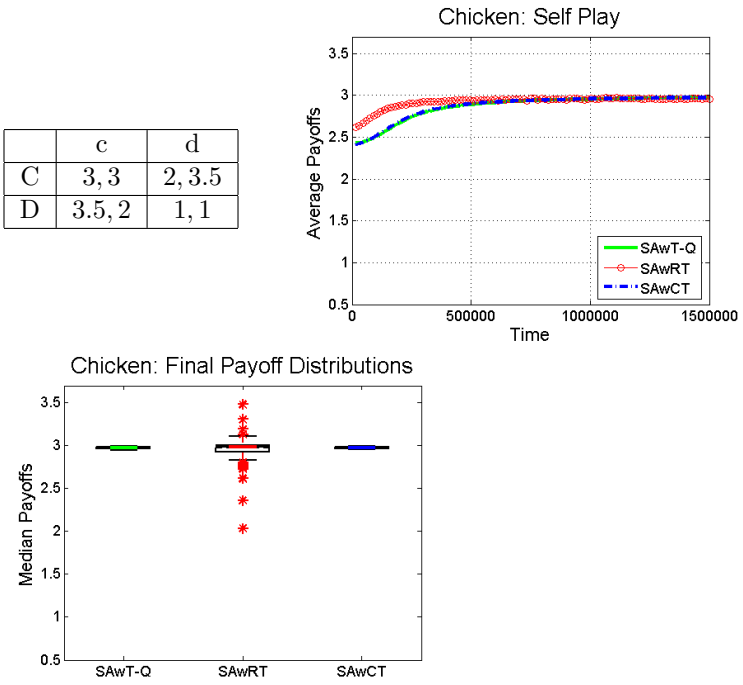


Figure 4.7: Results from 50 trials of the matrix game chicken.

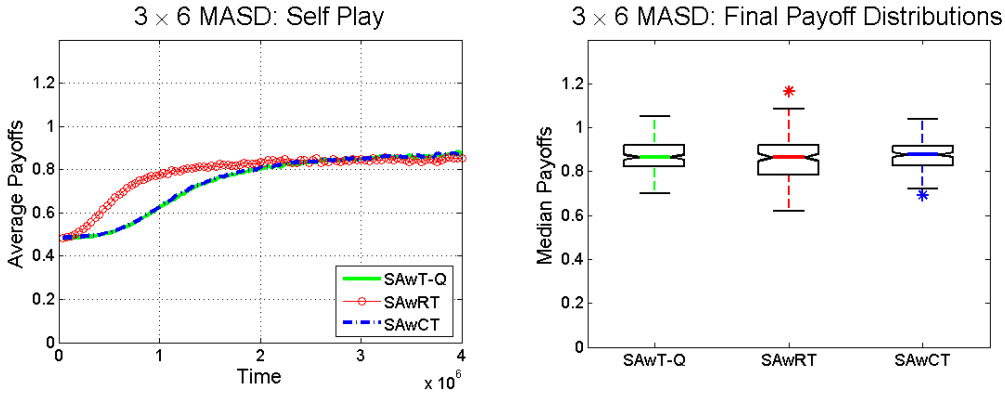


Figure 4.8: Results from 50 trials of a 3-agent, 6-action MASD (k=0.6). For these results $\lambda = 0.995$.

that naive implementations of SAwT do not tremble their aspirations to satisfy the conditions of pareto efficiency (see Figure 4.4) in this game. As a result, they learn to play pareto-dominated solutions, which results in inferior average payoffs to the agents (in this game). However, SAwT-Q does learn to tremble its aspirations to satisfy these conditions. Thus, it learns to play the PE solution (B, b) in this game. As SAwT-Q learns, its performance becomes similar to that of SAwT-Fixed, which always trembles its aspirations to the value $(3.99, 2.99)$. Notice that random aspiration trembles to the tremble values in $\mathcal{C}_i$ do not produce this result.

A Game with a Preclusion Critical Event

Figure 4.10 shows a 3×3 matrix game with a preclusion critical event when the aspiration levels of both agents are 3 or less. In this game, SAwT-Q learns to tremble its aspirations so that it avoids this preclusion critical event as well as maximize its payoffs. It learns to tremble its aspirations as does SAwCT. Thus, its performance becomes similar to that of SAwCT. Notice, again, that random trembles do not produce these same performance levels. However, SAwRT's performance is not bad in this game since, again, many of its trembles are in the downward direction when its aspiration levels are high.

4.5 SAwT in Changing Environments

In both this chapter and the previous chapter, we have assumed a stationary payoff matrix. However, in many situations (such as multi-state stochastic games, which we discuss in Chapter 6), the payoff matrix is non-stationary.

The S-Algorithm is not likely to perform well in games with a non-stationary payoff matrix. One of the main reasons that it will not perform well is that it uses a relaxation search (starting at a high value) to find a profitable solution that all agents can agree on (are satisfied with). However, the relaxation search considers only the current payoffs. Thus, once payoffs change (particularly in the upward direction), aspirations will no longer be high (with respect to the payoffs of the game). Thus, any subsequent relaxation searches (initiated by the agents becoming dissatisfied because of changing payoffs) have no guarantees of converging to a profitable strategy with high probability.

However, (large) aspiration trembles allow a satisficing agent to begin relaxation searches at high values at the beginning of most epochs. Thus, SAwT will continue to learn to play ε-PE solutions as long as the payoff matrix does not change too quickly. The speed at which the payoff matrix can change to maintain this result is dependent on the parameters λ (which controls the length of the relaxation search) and η. As λ gets bigger and as η gets smaller, the environment must necessarily change slower if SAwT is to play ε-PE solutions most of the time. Note, however, that decreasing λ and increasing η results in the agent playing ε-PE solutions less frequently when

	a	b
A	$3.5, 2$	$3.5, 1$
B	$1, 4$	$4, 3$

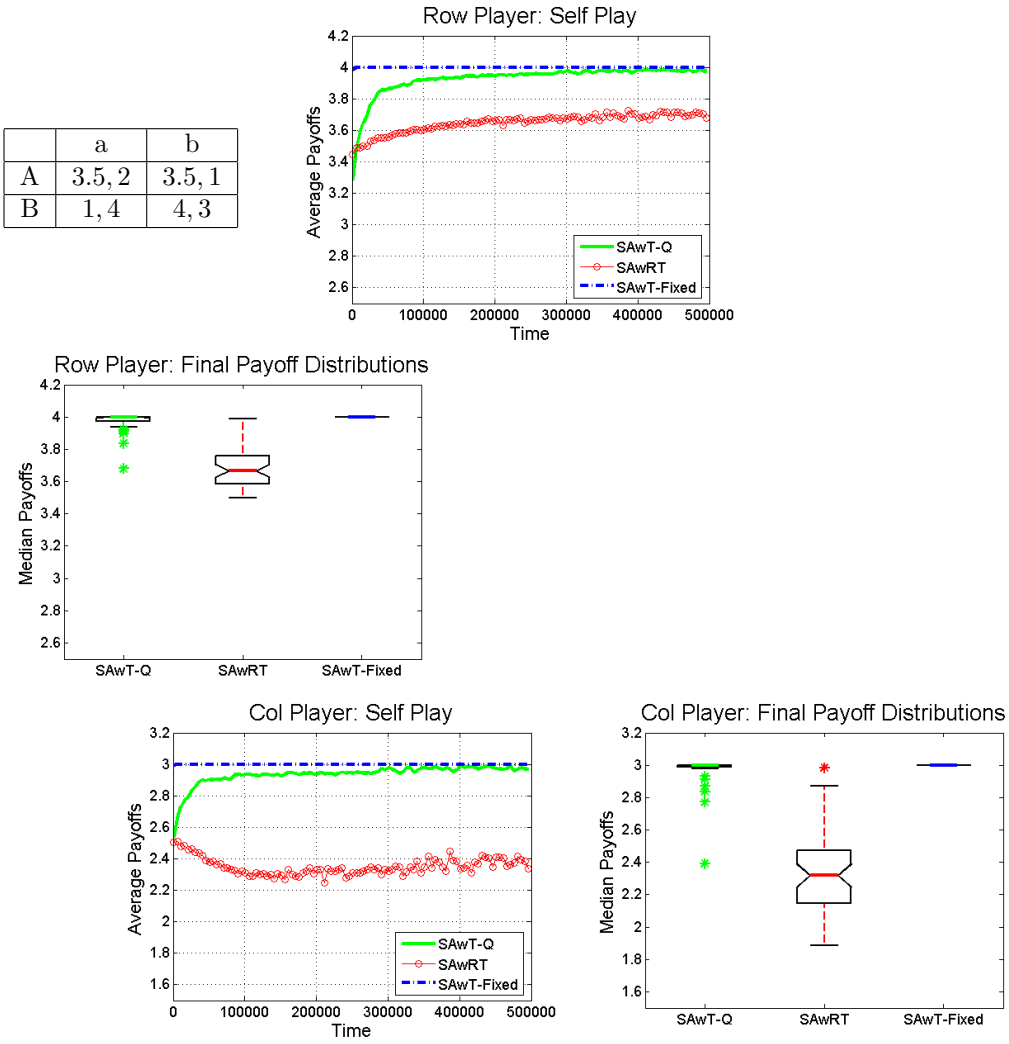


Figure 4.9: Results from 50 trials of the game shown at left. The performance of the row player is shown above and the performance of the column player is shown below. The SAwT implementation SAwT-Fixed is an agent that always trembles to the point $(3.99, 2.99)$.

	a	b	c
A	$0, 3$	$3, 0.5$	$3, 0$
B	$0.5, 3$	$3.3, 3.3$	$1.5, 3$
C	$3, 1.5$	$3, 1$	$1, 3$

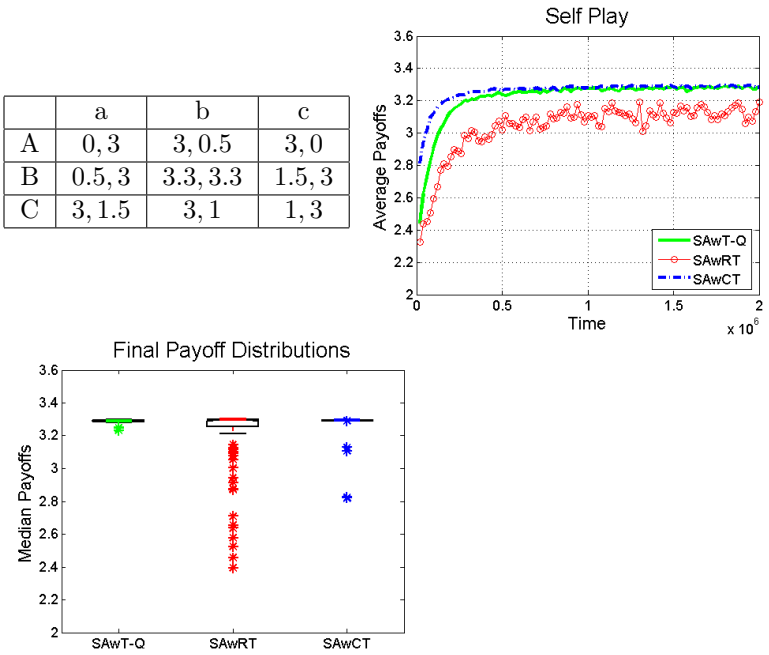


Figure 4.10: Results from 50 trials of the game shown at left. A learning rate of $\lambda = 0.995$ was used.

the payoff matrix is stationary since a lower λ means less likely convergence to ε-PE solutions and a higher η means a shorter convergence period. Thus, simply decreasing λ or increasing η to adapt to a changing payoff matrix will not typically lead to better performance

We note that the ability to adapt to changing payoff matrices is not a common attribute of multiagent learning algorithms (to date), although there are some exceptions (see, for example, [15]). Algorithms that depend on learned models (particularly those that are learned slowly) have a difficult time adapting to changing payoffs. Often these algorithms can never overcome the biases created by prior learning. One example of this is fictitious play [33] which learns different strategies depending on the agent's prior beliefs about their associates. When a payoff matrix changes over time, strategies learned under prior matrix configurations can result in permanent biases, which can result in undesirable behavior.

With the exception of the models of $r_i^{\max}$ and m_i^P (modeling which could be easily implemented for changing payoff matrices), naive implementations of SAwT do not use models to govern behavior. Thus, a slowly changing payoff matrix does not alter how it learns, as SAwT essentially restarts the learning process anew each time it trembles its aspirations (particularly in the case of large aspiration trembles). We note, however, that SAwT-Q learns a model that is not likely to adapt quickly enough to changing payoff matrices, especially when η decreases over time.

While SAwT promises to be effective under some changing conditions, it is limited by its minimal knowledge assumptions. We show in Chapter 6 how an algorithm similar to SAwT (presented in Chapter 5) can be extended to perform well in stochastic games. This algorithm requires knowledge of the actions of the other agents in the game so that an agent can observe its own payoff matrix.

4.6 Summary, Discussion, and Future Work

In this chapter, we extended the S-Algorithm to include aspiration trembles. The resulting algorithm SAwT has a number of desirable properties. First, SAwT removes the restriction that initial aspirations be made high. Second, SAwT agents can learn to tremble aspirations so that the resulting joint aspirations satisfy the conditions of pareto efficiency in many games. This results in higher average payoffs to the agents (see, for example, Figures 4.9 and 4.10).

Also, SAwT shows promise for continuing to play ε-PE solutions even in games in which the payoff matrix changes slowly over time. This result lays the foundation for *learning restarts*, which are used by a similar algorithm that we present in Chapters 5 and 6.

Future work could include a more in-depth analysis of tremble matrix games. In this chapter, we assumed that single agent Q-learners can learn this game effectively. In the games we analyzed, this algorithm performed effectively. However, it is possible that Q-learning will not be able to learn the tremble matrix game effectively in some games.

Chapter 5

Learning to Compete, Compromise, and Cooperate in Repeated General-Sum Matrix Games Using Reinforcement Learning

5.1 Introduction

In the previous chapter, we assumed that an agent did not know the actions and payoffs of other agents in the environment. While this is a reasonable knowledge restriction for many important domains, many other domains allow an agent to view the actions taken by other agents in the environment. Thus, in this chapter, we assume that an agent can view the actions of all agents in the environment and its own payoffs, but not the payoff of other agents. Thus, agent i knows R_i, but not R_j, for all $j \neq i$.

Even with knowledge of the actions of the other agents, learning successfully in repeated general-sum matrix games is a difficult problem. To see this, consider Figure 5.1. Figure 5.1(a) shows R_1, the payoff matrix for the row player. This payoff matrix, however, does little to tell the row agent how to play since R_1 does nothing to indicate the kind of game the agent is playing. For example, Figure 5.1(b)–(d) give examples of three different kinds of games which the agent could be playing; (b) shows a competitive game, (c) shows a fully cooperative game, and (d) shows a game of conflicting interest.

Although knowing one's own payoff matrix does not indicate the kind of game

	a	b
a	1	-2
b	-3	4

(a)

	a	b
a	1, -1	-2, 2
b	-3, 3	4, -4

(b)

	a	b
a	1, 1	-2, -2
b	-3, -3	4, 4

(c)

	a	b
a	1, 4	-2, -3
b	-3, -2	4, 1

(d)

Figure 5.1: Example showing how knowledge of one's own payoff matrix R_i does not indicate the kind of game an agent is playing. (a) Shows player 1's payoff matrix R_1, and (b)–(d) show three possible games the agent could be playing (given this payoff matrix).

an agent is playing, knowing the actions of other agents does provide valuable information. First, when an agent knows its own payoff matrix R_i, it can calculate its minimax value over the mixed strategy space[1]. The minimax value can serve as a lower bound on the average payoff that an agent should expect to receive. Second, the agents can use the previous ω joint actions as a signal that can be used to coordinate profitable compromises. Third, knowing the payoff matrix allows an agent to learn successfully in stochastic environments.

The first and second advantages just listed provide the foundation for (at least) two desirable properties of learning agents. First, an agent should never learn to play a strategy that produces expected payoffs lower than its minimax value. This property is particularly relevant in competitive games. Second, an agent should learn to make and accept compromises that increase its average payoffs. Thus, in cooperative games and in games of conflicting interest, an agent should learn to influence its associates so that its own payoffs are significantly greater than its minimax value. Though these two properties are simple and fundamental, no learning algorithm in the literature is known to possess them both (simultaneously) in general-sum games without knowing the payoffs of other agents (until preliminary results of this chapter appeared in [20]).

In this chapter, we present a reinforcement learning algorithm (called M-Qubed) that provably satisfies the first property in all general-sum matrix games (regardless of the associate) and empirically displays (in self play) the second property in a wide range of general-sum games.

5.2 Related Work

In this chapter, we present minimum criteria (in the form of two properties) that multiagent learning algorithms should satisfy (at the least) in repeated general-sum matrix games. Other minimum criteria have been suggested in the literature. One such set of criterion, given by Bowling and Veloso in [9], is that an agent's strategy should a) necessarily converge and b) converge to a best-response when associates' strategies converge. Bowling and Veloso presented an algorithm that provably satisfies this criteria in repeated 2×2 matrix games [9]. Conitzer and Sandholm gave an algorithm in [18] that provably satisfies this criteria in all repeated general-sum matrix games.

Another criterion used for evaluating multiagent learning algorithms is no-regret, or universal consistency [31, 33, 8]. If one of the experts used by a no-regret algorithm is the agent's minimax strategy, then no-regret learning guarantees that an agent will not learn to play a strategy that yields an average payoff below its minimax value. As we mentioned in Chapter 2, no-regret algorithms do not consider how an agent's

[1]Note that in the previous chapter we used the minimax value m_i^P calculated over the pure strategy space. The minimax value over the mixed strategy space (m_i) provides the agent with much more security in many games.

actions affect the future actions of associates. As a result, algorithms designed to meet this criteria learn myopic strategies in many theoretically important repeated games.

Yet another set of criterion for multiagent learning algorithms is given by Powers and Shoham [63, 62][2]. In these works, they propose three criteria that are similar in nature to the two criteria we advocate in this chapter (as well as in [20]). Powers *et al.*'s criteria are more specific (and less general) but tighter then the criteria presented herein.

The reinforcement learning algorithm we present in this chapter is an independent action learner [16], but it uses the previous joint actions of the agents as state. This idea was first used in Q-learning by Sandholm and Crites [66]. Additionally, it has been advocated by Moody [57] and used effectively in [20, 6] (among other places).

This work relies heavily on work done in reinforcement learning, which we reviewed briefly in Chapter 2. Thorough reviews of reinforcement learning can be found in [44, 75].

5.3 Terminology

In this section, we give some definitions and notation that we use in this chapter. We note that the terminology and notation presented in the section differs in some significant points from the terminology and notation used in the previous chapters.

Let S denote the set of states in the game and let A_i denote the set of actions that agent/player i may select in each state $s \in S$. Let $\mathbf{a}^t = (a_1^t, a_2^t, \ldots, a_n^t)$, where $a_i^t \in A_i$, be the *joint action* taken by the n agents on episode t, and let $A = A_1 \times \cdots \times A_n$ be the set of possible joint actions.

A *strategy* (or *policy*) for agent i is a probability distribution $\pi_i(\cdot)$ over its action set A_i. Let $\pi_i(S)$ denote a strategy over all states $s \in S$ and let $\pi_i(s)$ (or π_i) denote a strategy in a single state s. A strategy may be a *pure strategy* (the agent selects an action deterministically) or a *mixed strategy* (otherwise). A *joint strategy* played by n agents is denoted by $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$. Also, let $\mathbf{a}_{-i}$ and $\boldsymbol{\pi}_{-i}$ refer to the joint action and strategy of all agents except agent i.

Central to game theory is the matrix game, which is defined by a set of matrices $R = \{R_1, \ldots, R_n\}$. Let $R_i(\mathbf{a})$ be player i's payoff distribution when the joint action $\mathbf{a} \in A$ is played, and let $\mu_i(\mathbf{a})$ be the expected payoff to player i when the joint action $\mathbf{a}$ is played (i.e., $\mu_i(\mathbf{a}) = E[R_i(\mathbf{a})]$, $E[\cdot]$ denotes an expected value). Also, let $\mu_i(\boldsymbol{\pi})$ be the expected payoff to player i, when the joint strategy $\boldsymbol{\pi}$ is played, and let $\mu_i(\pi_i, \boldsymbol{\pi}_{-i})$ be the expected payoff to player i when it plays π_i and when the the other agents play the joint strategy $\boldsymbol{\pi}_{-i}$. Let $\mu_i^{\max} = \max_{\mathbf{a} \in A} \mu_i(\mathbf{a})$ and $\mu_i^{\min} = \min_{\mathbf{a} \in A} \mu_i(\mathbf{a})$

[2] In [62], Powers and Shoham give a slightly more in-depth review of past standards (i.e., minimum criteria) for multiagent learning that have been used in the literature. Our discussion of past minimum criteria follows their review closely.

Each matrix game has certain game theoretic values. The *minimax value* for agent i (denoted m_i) is

$$m_i = \max_{\pi_i} \min_{\mathbf{a}_{-i}} \mu_i(\pi_i, \mathbf{a}_{-i}) \tag{5.1}$$

The *minimax strategy* for agent i (denoted π_i^m) is

$$\pi_i^m = \operatorname*{argmax}_{\pi_i} \min_{\mathbf{a}_{-i}} \mu_i(\pi_i, \mathbf{a}_{-i}) \tag{5.2}$$

A *one-shot NE* is a joint strategy $\boldsymbol{\pi}$ such that no agent can unilaterally change its strategy without lowering its expected payoff in the current episode. Nash [60] showed that every n-player matrix game has at least one one-shot NE. An *NE of the repeated game* (rNE) is a joint strategy $\boldsymbol{\pi}(S)$ over all states $s \in S$, such that no agent can unilaterally change its strategy in any (reachable) state $s \in S$ without lowering its expected average payoff over time. Every one-shot NE is an rNE.

We assume that an agent can observe its own payoffs as well as the actions taken by all agents. Thus, an agent can observe its own payoff matrix R_i. This being the case, we assume in this chapter that a) an agent knows its payoff matrix R_i (or at least $\mu_i(\mathbf{a})$ for all $\mathbf{a} \in A$) prior to the beginning of play (and, therefore, can calculate its minimax value m_i and minimax strategy π_i^m using linear programming) and b) $R_i(\mathbf{a})$ is stationary for all $\mathbf{a} \in A$.

5.4 Minimum Criteria

As we mentioned in the introduction of this dissertation, the folk theorem [35] implies that, in many games, there exist rNEs that yield higher payoffs to all agents than do one-shot NEs. Thus, the one-shot NE should not be chosen as the goal of learning agents. Rather, in repeated games between successful learning agents, a successful agent should learn to play profitable rNEs. However, it is not clear which rNE an agent should learn to play. Additionally, in games between learning agents, it is difficult for a learner to determine if it is playing its portion of an rNE. Because of this, we seek to design an agent that learns to play strategies that yield average payoffs that correspond to an rNE, preferably an rNE with a high payoff. Thus, a learner should possess (at the least) the following two properties.

Property 5.1. *(Security Property) An agent should not learn to play strategies that produce average expected payoffs below its minimax value m_i.*

An agent can easily satisfy this property by playing π_i^m (the minimax strategy) regardless of what its associates do. However, such a strategy may be "irrational," since m_i (the minimax value) may be much lower than the payoffs that an agent can realistically expect. On the other hand, m_i is the highest expected payoff that any agent can guarantee itself without some form of cooperation or compromise from

its associate(s). Thus, a successful agent must learn (if possible) to influence its associate(s) to be cooperative. This brings us to the second property, which we define next (after introducing the necessary terminology).

Let $\Omega_i^t(\boldsymbol{\pi}_{-i}^t(S)) = \{\pi_i(S) : \mu_i(\pi_i(S), \boldsymbol{\pi}_{-i}^t(S)) > m_i\}$ and let $\Delta_{-i}(S) = \{\boldsymbol{\pi}_{-i}(S) : \Omega_i(\boldsymbol{\pi}_{-i}(S)) \neq \emptyset\}$. That is, $\Omega_i^t(\boldsymbol{\pi}_{-i}^t(S))$ is the set of all strategies for agent i (at time t) that will give it an expected payoff greater than its minimax value given the other agents' strategies $\boldsymbol{\pi}_{-i}^t(S)$; and $\Delta_{-i}(S)$ is the set of the agent's associates' strategies for which Ω_i^t is not empty. Also, let $\varphi_i^T = (a_i^0, a_i^1, \ldots a_i^T)$ be an action sequence through time T for agent i, where a_i^t is agent i's action at time t. Let $\Phi_i = \{\varphi_i^T : \forall t \geq T, \boldsymbol{\pi}_{-i}^t(S) \in \Delta_{-i}(S)$ if, $\forall \tau \in [T, t), \pi_i^\tau(S) \in \Omega_i^\tau(\boldsymbol{\pi}_{-i}^\tau(S))\}$. That is, Φ_i is the set of action sequences that agent i may take that lead to it receiving an expected payoff greater than its minimax value from time T onward (provided that it continues to play appropriately).

Property 5.2. *(Compromise/Cooperate Property) This property has two parts:* influence *and* compromise.

- Influence. *If $\Phi_i \neq \emptyset$, then an agent should play $\varphi_i \in \Phi_i$ prior to time T and $\pi_i^t(S) \in \Omega_i^t(\boldsymbol{\pi}_{-i}^t(S))$ thereafter.*
- Compromise. *An agent can be* influenced *in non-competitive games (see Section 2.5).*

We refer to this property as the *compromise/cooperate (C/C) property* because an agent must offer and accept compromises to possess it. In words, *influence* says that if an agent can influence its associate(s) to play cooperatively, it will. *Compromise* says that an agent can be induced to cooperate/compromise.

It has been shown in [27] that an agent without omniscience cannot possess the *C/C* property when associating with all associates. However, a successful learning algorithm should possess this property when associating with a large class of agents. In this chapter, we focus on the *C/C property* in self play only.

Note that the definition of the *C/C property* means that an algorithm satisfies this property when $\mu_i(\pi_i(S), \boldsymbol{\pi}_{-i}(S)) = m_i + \epsilon$, where $\epsilon > 0$. This is hardly useful since ϵ is allowed to be very small, but defining a threshold greater than m_i is difficult for the general case. Ideally, we argue that when *intelligent* learning agents associate in repeated games, the learned joint strategies should be pareto efficient. However, a self-interested agent is not concerned with the payoffs of others (unless the payoffs of others affect its own future payoffs). Thus, an algorithm could possibly satisfy the *C/C property* without learning to play a pareto efficient solution.

To remedy this dilemma, we advocate that the *C/C property* can be satisfied to various degrees. For example, suppose in a given game, that both agent A and agent B satisfy the *C/C property* in self play. We say that agent A more fully satisfies the *C/C property* in this game (in self play) than does agent B if $\mu_A(\pi_A(S), \boldsymbol{\pi}_{-A}(S)) > \mu_B(\pi_B(S), \boldsymbol{\pi}_{-B}(S))$.

Learning Algorithm	Security Property	C/C Property (Self Play)
Q-Learning	No	No
Fictitious Play [33]	No	No
minimaxQ [51]	Yes	No
NashQ [41]	No	No
FFQ [52]	Yes	Only in cooperative games
WoLF-PHC [9]	No	No
Satisficing Learning [71, 24]	No	In most noncompetitive games
GIGA-WoLF [8]	Yes	No

Table 5.1: Property characteristics of typical learners in repeated general-sum games. Note that there are many algorithms that have been studied in the context of learning to cooperate in small classes of games. We do not list them here to emphasize the focus on general-sum games as a whole.

While these two properties should be considered simple and basic to the multiagent learning problem, until preliminary results of this chapter appeared in [20], no known learning algorithm in the literature was known to possess both of these properties simultaneously in repeated general-sum matrix games (even in self play) without having full knowledge of the complete game matrix R[3]. Table 5.1 gives a short representative list of algorithms in the literature and indicates which of the two properties that they possess.

5.5 The M-Qubed Algorithm

In this section, we describe a learning algorithm that (provably) satisfies the *security* property and appears (empirically) to possess the *C/C* property in self play. We call this algorithm M-Qubed (*M*ax or *M*ini*m*ax *Q*-learning = M^3-Q = M-Qubed).

Like Q-learning [76], M-Qubed estimates the value of a state-action pair and selects a strategy using this estimate. We now describe the algorithm.

5.5.1 Q-update

In Q-learning, the quality of a state-action pair $Q(s, a_i)$ is estimated by iteratively updating Q-values using the following equation:

$$Q_i^{t+1}(s, a_i^t) = (1 - \alpha_i)Q_i^t(s, a_i^t) + \alpha_i \left(r_i^t + \gamma_i V_i^t(s')\right) \tag{5.3}$$

[3]Powers and Shoham (in [62]) state a similar set of properties and give an algorithm that possesses these properties when associating with a certain class of agents. However, their algorithm requires complete knowledge of R.

where s is the current state, a_i is the action taken at time t, r_i^t is the reward receive at time t, and $V_i^t(s')$ is the estimated value of the next state s', given by

$$V_i^t(s') = \max_{b_i} Q_i^t(s', b_i) \tag{5.4}$$

Watkins showed that the estimate $Q_i^t(s, a_i)$ approaches the true Q-values as $t \to \infty$ in stationary environments provided that each state-action pair is taken infinitely often and the learning rate α_i is decreased appropriately [76]. However, since matrix games between learning agents effectually produce non-stationary environments, various algorithms have been proposed to make Q-learning appropriate for these situations. These approaches involve a) estimating the value of a state-joint action pair (rather than a state-action pair) and b) estimating the value of the next state $V_i^t(s')$ with a game theoretic value (e.g., [51, 41, 52, 39]).

We note that while matrix games do not have state, agents can benefit from using the previous w joint actions taken by the agents as state (see [66]), provided that the actions of all agents can be observed (which we assume). Let $H(\omega_i)$ be the set of all histories of joint actions of length ω_i. Then, in matrix games, $S = H(\omega_i)$ and $s = h^t(\omega_i) \in H(\omega_i)$, where $h^t(\omega_i) = (\mathbf{a}^{t-\omega_i}, \cdots, \mathbf{a}^{t-1})$.

An additional observation about Q-updates in matrix games in which the payoff matrix is stationary is that the expected reward $\mu_i(\mathbf{a}^t)$ can be used in place of the reward r_i^t. This is because r_i^t is a sample taken from the distribution $R_i(\mathbf{a}^t)$. While this substitution really does nothing in theory, it does in practice provide added stability (and can increase the speed of learning) when $R_i(\mathbf{a})$ has high variance.

These observations can be used to transform Equation (5.3) to

$$Q_i^{t+1}(h^t(\omega_i), a_i^t) = (1 - \alpha_i) Q_t(h^t(\omega_i), a_i^t) + \alpha_i \left[\mu_i(\mathbf{a}^t) + \gamma_i V_i^t(h^{t+1}(\omega_i))\right] \tag{5.5}$$

M-Qubed estimates the value of the next state $V_i^t(h^{t+1}(\omega_i))$ using

$$V_i^t(h^{t+1}(\omega_i)) = \sum_{b_i} \pi_i^t(h^{t+1}(\omega_i), b_i) Q_i^t(h^{t+1}(\omega_i), b_i) \tag{5.6}$$

where $\pi_i^t(h^{t+1}(\omega_i), b_i)$ is the probability that the agent believes it will play action b_i in state $h^{t+1}(\omega_i)$.[4] Note that M-Qubed's estimated value of the next state given in Equation (5.6) is equivalent to the one used in Q-learning (see Equation (5.4)) if $\pi_i^t(h^{t+1}(\omega_i)) = \text{argmax}_{b_i} Q_i^t(h^{t+1}(\omega_i), b_i)$.

5.5.2 Selecting a Strategy

A Q-learner always (unless exploring) takes the action corresponding to its highest Q-value in the current state. However, in repeated matrix games, an agent's best

[4]Equation (5.6) makes M-Qubed's Q-update similar to the on-policy learning method called Sarsa [75].

strategy might not be a pure strategy. The trick is to know when to play a pure strategy and when to play a mixed strategy. We advocate that agents should act predictably (pure strategies) in games of compromise and cooperation, but unpredictably (mixed strategies) in competitive games (if π_i^m is a mixed strategy). Since it is frequently unknown *a priori* whether a game is competitive or whether associates will be inclined to cooperate, a successful learning algorithm must determine whether to act predictably or unpredictably.

In this subsection, we state two strategy rules for determining whether M-Qubed should play predictably or unpredictably. Since each strategy rule is advantageous for a different set of circumstances, we present a third strategy rule that mixes the two rules in a desirable fashion.

In the first strategy rule, M-Qubed determines whether it should play predictably or unpredictably by comparing its highest Q-value in the current state ($\max_{a_i} Q_i^t(h^t(\omega_i), a_i)$) with the discounted sum of m_i ($\sum_{t=0}^{\infty} {\gamma_i}^t m_i = \frac{m_i}{1-\gamma_i}$). When $\max_{a_i} Q_i^t(h^t(\omega_i), a_i) > \frac{m_i}{1-\gamma_i}$, M-Qubed plays predictably. Otherwise, it plays the minimax strategy π_i^m. Thus, M-Qubed's strategy in state $h^t(\omega_i)$ is

$$\pi_i^*(h^t(\omega_i)) \leftarrow \begin{cases} \text{argmax}_{a_i} Q_i^t(h^t(\omega_i), a_i) & \text{if } \max_{a_i} Q_i^t(h^t(\omega_i), a_i) > \frac{m_i}{1-\gamma_i} \\ \pi_i^m & \text{otherwise} \end{cases} \tag{5.7}$$

We call this strategy rule *Strategy Rule 1*. Unfortunately, an agent may be exploited when using this strategy rule since its Q-values might not reflect its actual payoffs when the environment is made non-stationary by the changing strategies of associates.

A second strategy rule (called *Strategy Rule 2*) for determining whether to play predictably or unpredictably is to compare the agent's average payoff $\mu_i^t = \frac{1}{t}\sum_{j=1}^{t} \mu_i(\mathbf{a}^j)$ with m_i. This gives

$$\pi_i^*(h^t(\omega_i)) \leftarrow \begin{cases} \text{argmax}_{a_i} Q_i^t(h^t\omega_i, a_i) & \text{if } \frac{\mu_i^t}{1-\gamma_i} > \frac{m_i}{1-\gamma_i} \\ \pi_i^m & \text{otherwise} \end{cases} \tag{5.8}$$

While safe, μ_i^t reacts too slowly. Thus, an agent using this strategy rule might a) continue to play π_i^m when a more profitable strategy can be learned otherwise or b) be slow to play π_i^m when it is being temporarily exploited.

Strategy Rule 1 allows an agent to explore possible improvements (C/C), while Strategy Rule 2 provides an agent with security. A balance may be obtained by learning which of the two techniques is more successful. Let β_i^t be the probability that M-Qubed uses Strategy Rule 2 at time t, and let $1 - \beta_i^t$ be the probability that it uses Strategy Rule 1. Thus,

$$\pi_i^*(h^t(\omega_i)) \leftarrow \begin{cases} \text{Strategy Rule 2} & \text{with probability } \beta_t \\ \text{Strategy Rule 1} & \text{with probability } 1 - \beta_t \end{cases} \tag{5.9}$$

We call this strategy rule *Strategy Rule 3*.

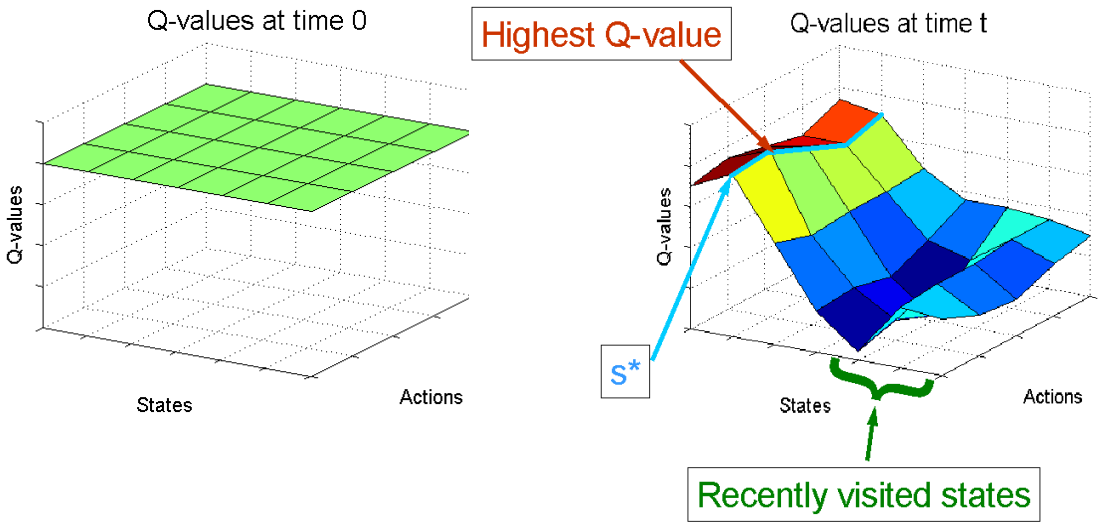


Figure 5.2: M-Qubed sets in is Q-values high initially (left) and randomizes its strategy with a small probability (η_i) when it has not visited the state containing the highest global Q-value recently (right).

β_i^t is determined using the intermediate variable $\acute{\beta}_i^t$. Initially, M-Qubed optimistically sets $\acute{\beta}_i^0 = 0$, then updates it using

$$\acute{\beta}_i^{t+1} \leftarrow \begin{cases} \acute{\beta}_i^t + \lambda(r_i^t - m_i) & \text{Strategy Rule 2} \\ \acute{\beta}_i^t - \lambda(r_i^t - m_i) & \text{Strategy Rule 1} \end{cases} \tag{5.10}$$

where λ is some constant. In words, $\acute{\beta}_i^t$ determines which rule is more successful by increasing (toward Strategy Rule 2) if Strategy Rule 2 yields payoffs greater than m_i or if Strategy Rule 1 yields payoffs lower than m_i, and decreasing (toward Strategy Rule 1) otherwise. β_i^t is $\acute{\beta}_i^t$ constrained to the interval $[0, 1]$, or

$$\beta_i^t = \min(1, \max(0, \acute{\beta}_i^t)) \tag{5.11}$$

5.5.3 Adding Exploration

Q-learners typically explore their environment using ε-greedy or Boltzmann exploration. In repeated matrix games, an alternate exploration method is to set initial value estimates optimistically (as suggested in [72]) and let the learning process dictate exploration. M-Qubed does this by initializing each Q-value to its highest possible expected discounted reward, $\frac{\mu_i^{\max}}{1-\gamma_i}$ (see Figure 5.2(left)). In this way, an agent uses a relaxation search to find a strategy that yields an expected discounted reward at least as high as its highest Q-value.

M-Qubed also randomizes its strategy with some small probability η_i if it perceives that its current strategy π_i^* keeps it in a local maximum. For example, consider the

hypothetical Q-values shown in Figure 5.2(right). In the figure, the highest global Q-value is in state s^*. However, only the rightmost states have been visited "recently," none of which have Q-values as high as does s^*. Thus, M-Qubed randomizes its strategy with probability η_i in hopes of entering state s^*[5].

M-Qubed guesses that its current strategy keeps it in a local maximum if none of the states which it has visited in the previous $w = (\prod_i |A_i|)^{\omega_i}$ (the number of states in the game) episodes of the game have Q-values *close* to the highest Q-value in state s^*. More formally, let $Q_i^{\max} = \max_{a_i \in A_i} Q_i^t(s^*, a_i)$ and let $H_{t-w}^{t-1} = \{h^{t-w}(\omega_i), \cdots, h^{t-1}(\omega_i)\}$. Then, M-Qubed's current strategy keeps it in a local maximum if $Q_i^{\max} > \max_{h \in H_{t-w}^{t-1}, a_i \in A_i} Q_i^t(h, a_i) + \varepsilon$, where ε is some small positive constant. Thus,

$$\pi_i^t(h^t(\omega_i), a_i) \leftarrow \begin{cases} \pi_i^*(h^t(\omega_i), a_i), & \text{if } Q_i^{\max} \leq \max_{h \in H_{t-w}^{t-1}, a_i \in A_i} Q_i^t(h, a_i) + \varepsilon \\ (1 - \eta_i)\pi_i^*(h^t(\omega_i), a_i) + \eta_i \frac{1}{|A_i|}, & \text{otherwise} \end{cases} \tag{5.12}$$

where $\pi_i^*(h^t(\omega_i), a_i)$ is obtained from Equation (5.9) (unless stated otherwise). η_i is slowly decayed to 0 as $t \to \infty$.

Because M-Qubed can cease to explore, it can learn strategies that are not as desirable as it otherwise might have learned. However, we will show in the next section that continuing to explore in repeated games can be more costly than ceasing exploration.

5.6 Results

In this section, we prove that M-Qubed (summarized in Algorithm 5.1) satisfies the security property and show that it displays the *C/C* property in a wide range of games (in self play). Unless stated otherwise, the algorithms use the parameter values shown in Table 5.2.

5.6.1 Security Property

Before giving the main theorem, we give two lemmas related to the random walk of $\acute{\beta}_i^t$ since its behavior relates closely to the performance of M-Qubed. The first lemma shows that the interval $(0, 1)$ is a reflecting boundary for $\acute{\beta}_i^t$ when M-Qubed is being exploited (it reflects $\acute{\beta}_i^t$ upward). The second lemma states a property of reflected random walks.

Changes in $\acute{\beta}_i^t$ are related to the losses of agent i. Let $l_i^t = (m_i - r_i^t)$ be agent i's loss at time t. Then, $\acute{\beta}_i^t$ is updated using (adapted from Equation (5.10))

$$\acute{\beta}_i^{t+1} \leftarrow \begin{cases} \acute{\beta}_i^t - \lambda l_i^t & \text{Strategy Rule 2} \\ \acute{\beta}_i^t + \lambda l_i^t & \text{Strategy Rule 1} \end{cases} \tag{5.13}$$

[5]Note, again, that s^* is simply the previous ω_i joint actions taken by the agents prior to the current time t. We use the term s^*, rather than $h^t(\omega_i)^*$ for notational simplicity.

1) $\alpha_i, \eta_i \in [0,1]$, $\acute{\beta}_i = \beta_i = 0$.
2) Initialize $\pi_i^t(h^t(\omega_i), a_i) \leftarrow \frac{1}{|A_i|}$, $Q(h^t(\omega_i), a_i) \leftarrow \frac{\mu_i^{\max}}{1-\gamma_i}$
3) $t \leftarrow 1$
4) Repeat,
    a) Take action a_i according to $\pi_i^t(h^t(\omega_i))$
    b) Observe reward r_i^t and joint action $\mathbf{a}^t$:
        i) Update $\pi_i^t(h^t(\omega_i))$ using Equation (5.12)
        ii) Update $Q_i^t(h^t(\omega_i), a_i)$ using Equations (5.5) and (5.6)
        iii) Update $\acute{\beta}_i$, β_i using Equation (5.10)
    c) $t \leftarrow t+1$

Algorithm 5.1: The M-Qubed algorithm for player i.

Name	**Type**	**Parameter details**
M-Qubed	M-Qubed	$\alpha_i = 0.1$, $\gamma_i = 0.95$, $\omega_i = 1$, $\varepsilon = \frac{0.005(\mu_i^{\max} - \mu_i^{\min})}{1-\gamma_i}$ $\eta_i = 0.04\left(\frac{20000}{20000+t}\right)$, $\lambda = \frac{0.01\alpha_i}{\mu_i^{\max} - \mu_i^{\min}}$
QL	Q-learner	$\alpha_i = 1/(10 + 0.01\kappa_s^{a_i})$, $\gamma_i = 0.95$, ε-greedy w/ $\varepsilon = \max(0.2 - 0.0006t, 0)$
rQL	Q-learner	$\alpha_i = 1/(10 + 0.01\kappa_s^{a_i})$, $\gamma_i = 0.95$, ε-greedy w/ $\varepsilon = 0.1$
WoLF	WoLF-PHC	Same as QL, but with $\delta_w = 1/(10.0 + \kappa_s^{a_i})$, $\delta_l = 4\delta_w$

Table 5.2: Learners and their parameter values. $\kappa_s^{a_i}$ is the number of times that action a_i has been played in state s.

Thus, when the agent receives a payoff less than its minimax value, $\acute{\beta}_i^t$ moves away from the strategy rule that was played during that episode. Recall that the closer $\acute{\beta}_i^t$ gets to 0 (or below) the more likely the agent is to play Strategy Rule 1, and the closer $\acute{\beta}_i^t$ gets to 1 (or above) the more likely the agent is to play Strategy Rule 2.

Let $L_i^t = \sum_{j=0}^{t-1} l_i^j$ be agent i's accumulated loss prior to time t, and let $[\tau, \rho]$ be an interval such that $\acute{\beta}_i^t \in (0,1)$ for all $t \in [\tau, \rho]$. Also, we say that agent i is being exploited if $\mu_i^t < m_i$.

Lemma 5.1. *$E[\acute{\beta}_i^\rho - \acute{\beta}_i^\tau] > 0$ if M-Qubed is being exploited during time $t \in [\tau, \rho]$.*

Proof. Let ξ_1 be the subset $t \in [\tau, \rho]$ in which M-Qubed uses Strategy Rule 1, and let ξ_2 be the subset $t \in [\tau, \rho]$ in which it uses Strategy Rule 2. Then M-Qubed's expected

loss during the time interval $[\tau, \rho]$ is

$$E[L_i^\rho - L_i^\tau] = \sum_{t=\tau}^{\rho} E[l_i^t] = \sum_{t \in \xi_1} E[l_i^t] + \sum_{t \in \xi_2} E[l_i^t] \tag{5.14}$$

When $\mu_i^t \leq m_i$, then $\sum_{t \in \xi_2} E[l_i^t] \leq 0$ since M-Qubed plays π_i^m.[6] That is, when agent i is being exploited, its expected loss when it uses Strategy Rule 2 is non-positive since it uses its minimax strategy. Thus, agent i's expected loss (when it is being exploited) is bounded by its expected losses during the times in which it plays Strategy Rule 1. Formally, agent i's expected loss is bounded (using Equation (5.14)) by $E[L_i^\rho - L_i^\tau] \leq \sum_{t \in \xi_1} E[l_i^t]$. Thus, M-Qubed can only have positive loss if $\sum_{t \in \xi_1} E[l_i^t] > 0$ when $\acute{\beta}_i^t \in (0, 1)$. Since $\acute{\beta}_i^t$ increases whenever M-Qubed has positive loss while playing Strategy Rule 1, if $E[L_i^\rho - L_i^\tau] > 0$, then $E[\acute{\beta}_i^\rho - \acute{\beta}_i^\tau] > 0$. Thus, if M-Qubed is exploited, then $E[\acute{\beta}_i^\rho - \acute{\beta}_i^\tau] > 0$ when $\acute{\beta}_i^t \in (0, 1)$ for all $t \in [\tau, \rho]$. □

Let W be a random walk in which the mean increases (i.e., $E[W_{t+1} - W_t] \geq 0$) for all W_t provided that it encounters no boundaries. Let there be a reflecting boundary such that whenever $W_t < b$, $E[W_{t+1} - W_t] = d$, where d is a positive constant. Also, let $Pr(\cdot)$ denote a probability.

Lemma 5.2. *$Pr(W_t < b) \to 0$ as $t \to \infty$.*

Proof. We consider the worst case, which occurs when $E[W_{t+1} - W_t] = 0$ whenever $W_t \geq b$. Since the process must increase (on average) by some positive constant d when $W_t < b$, it follows that $E[W_{t+1} - W_t] = d \cdot Pr(W_t < b) > 0$ (for all t). This must mean that for all t and for all c, $Pr(W_{t+T} < c) < Pr(W_t < c)$ for all $T > \tau$, where $\tau > 0$. Thus, we know that $f(t) = Pr(W_t < b)$ decreases as $t \to \infty$. But does $Pr(W_t < b)$ asymptotically approach some value greater than 0 as $t \to 0$? Suppose that it does. Then $\lim_{t \to \infty} Pr(W_t < b) \to a$, for some $a > 0$. But, since the difference between $Pr(W_{t+1} < b)$ and $Pr(W_t < b)$ is positively correlated with $E[W_{t+1} - W_t]$, this requires that $\lim_{t \to \infty} E[W_{t+1} - W_t] \to 0$. However, since $E[W_{t+1} - W_t] = d \cdot a$, $E[W_{t+1} - W_t] \to 0$ only if $a \to 0$ (since d is a positive constant). Thus, $Pr(W_t < b) \to 0$ as $t \to \infty$. □

We are now ready for the main theorem.

Theorem 5.1. *M-Qubed satisfies the security property (in the limit) in all repeated general-sum matrix games, regardless of its associate(s).*

Proof. We will show that $\lim_{t \to \infty} L_i^t / t \leq 0$ in the worst case. Note that this implies security since $\lim_{t \to \infty} \mu_i^t / t \geq m_i$ when $\lim_{t \to \infty} L_i^t / t \leq 0$.

[6] Since $\eta_i \to 0$ as $t \to \infty$ we can assume $\eta_i = 0$.

The worst case can occur when playing against an omniscient opponent, a *seer agent*, that knows everything about M-Qubed including its strategy, but not its actual actions (*a priori*).

If $L_i^t \leq 0$ for all $t \geq T$ (i.e., $\mu_i^t \geq m_i$ for all $t \geq T$), then $\lim_{t\to\infty} L_i^t/t \leq 0$. Otherwise, we must address three cases when $\mu_i^t < m_i$ (when the agent is being exploited).

Case 1: $\acute{\beta}_i^t \leq 0$ for all $t \geq T$. Since $\beta_i^t = 0$ when $\acute{\beta}_i^t \leq 0$, M-Qubed uses Strategy Rule 1 with probability 1. Thus, Equation (5.10) assures that $\acute{\beta}_i^t = \acute{\beta}_i^T - \lambda(L_i^T - L_i^t)$, so agent i's loss during $t \geq T$ is $L_i^t - L_i^T = \frac{\acute{\beta}_i^t - \acute{\beta}_i^T}{\lambda}$. Since $\acute{\beta}_i^t \leq 0$, $L_i^t - L_i^T \leq \frac{-\acute{\beta}_i^T}{\lambda}$ (a constant), so $\lim_{t\to\infty} L_i^t/t \leq 0$.

Case 2: $\acute{\beta}_i^t \geq 1$ for all $t \geq T$. Using similar logic as in case 1 (using Strategy Rule 2 rather than Strategy Rule 1), $\lim_{t\to\infty} L_i^t/t \leq 0$.

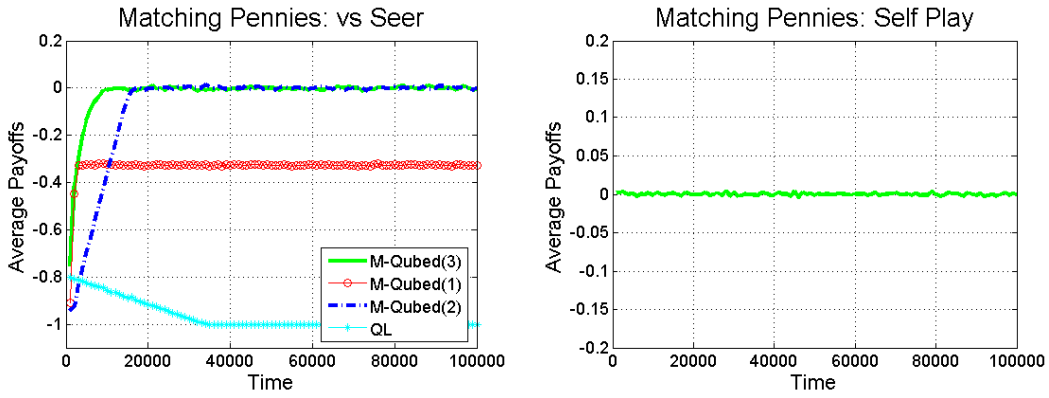
Case 3: $\acute{\beta}_i^t \in (0,1)$ for all t in some arbitrary time interval(s) $[\tau, \rho]$. Lemma 5.1 shows that $E[\acute{\beta}_i^\rho - \acute{\beta}_i^\tau]$ is proportional to M-Qubed's losses during the time $t \in [\tau, \rho]$. This means two things. First, a seer agent cannot increase M-Qubed's expected accumulated loss by more than a constant when $\acute{\beta}_i^t \in (0,1)$. Second (in conjunction with cases 1 and 2), M-Qubed's expected accumulated loss can increase by no more than a constant except when $\acute{\beta}_i^t$ repeatedly crosses 0 or 1. Each time $\acute{\beta}_i^t$ crosses 0 or 1, M-Qubed can experience a small loss (bounded by $(m_i - \mu_i^{\min})$). However, since M-Qubed uses Strategy Rule 1 when $\acute{\beta}_i^t \leq 0$, $\acute{\beta}_i^t \leq 0$ is a reflecting boundary. Since $\acute{\beta}_i^t$ also satisfies the preconditions of Lemma 5.2 it follows from Lemma 5.2 that $Pr(\acute{\beta}_i^t < 0) \to 0$ as $t \to \infty$. Thus, $\acute{\beta}_i^t$ crosses 0 increasingly less frequently as $t \to \infty$. A similar argument can be made for $\acute{\beta}_i^t$ crossing 1 since Lemma 5.1 shows that $(0,1)$ is also a reflecting boundary (thus, $\acute{\beta}_i^t < 1$ is a reflecting boundary). Therefore, $\lim_{t\to\infty} L_i^t/t \leq 0$. □

Figure 5.3(a) shows the average payoffs of various learners against a seer agent in the game Matching Pennies, shown in Figure 5.4(a). Since a Q-learner (unless exploring) plays the action corresponding to its max Q-value, a seer agent exploits it indefinitely. Also, if M-Qubed uses only Strategy Rule 1 (labeled M-Qubed(1)), then it may be exploited somewhat. However, if M-Qubed uses Strategy Rules 2[7] or 3 (labeled M-Qubed(2) and M-Qubed(3), respectively), it learns to play π_i^m and, therefore, cannot be exploited. Figure 5.3(b) shows M-Qubed's average payoffs in self play.

5.6.2 Cooperate and Compromise Property

In this subsection, we present empirical results showing that M-Qubed (in self play) displays the C/C property in many general-sum matrix games. In doing so, we compare its performance with the self play of other learning agents (see Table 5.2) in

[7]In Figure 5.3(a), m_i is replaced by the function $f(t) = \min(\mu_i^{\min} + \frac{t(m_i - \mu_i^{\min})}{30000}, m_i)$.



(a) (b)

Figure 5.3: (a) Average payoffs over time in the game matching pennies against a *seer* agent. Results are an average of 50 trials. Note that for M-Qubed(2), m_i is replaced by the function $f(t) = \min(\mu_i^{\min} + \frac{t(m_i - \mu_i^{\min})}{30000}, m_i)$ (see Equation (5.8)). (b) Average payoffs over time in the game matching pennies in self play. Results are an average of 50 trials.

	a	b
a	1, -1	-1, 1
b	-1, 1	1, -1

(a) Matching Pennies

	c	d
c	3, 3	0, 5
d	5, 0	1, 1

(b) Prisoner's Dilemma

	c	d
c	3, 3	2, 3.5
d	3.5, 2	1, 1

(c) Chicken

	a	b	c
a	0, 0	0, 1	1, 0
b	1, 0	0, 0	0, 1
c	0, 1	1, 0	0, 0

(d) Shapley's Game

	c	d
c	4, 4	-5, 3
d	3, -5	2, 2

(e) Staghunt

	a	b
a	0, 3	3, 2
b	1, 0	2, 1

(f) Tricky Game

Figure 5.4: Various 2-agent matrix games.

a number of games. We also show that M-Qubed does not always display the C/C property in some games when associating with other kinds of learners. Additionally, we compare the M-Qubed algorithm with aspiration-based satisficing learning.

Self Play

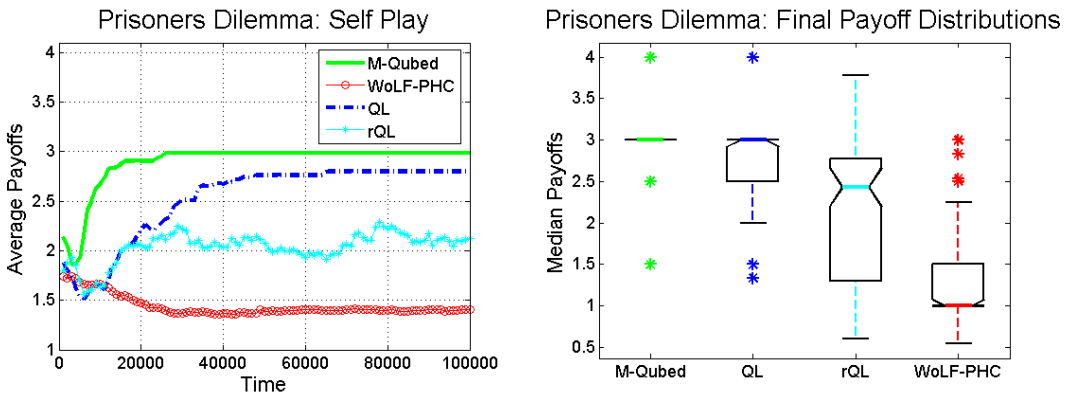
We compare the performance (in self play) of four reinforcement learning algorithms shown in Table 5.2 in the following games: the iterated prisoner's dilemma (see Figure 5.4(b)), chicken (see Figure 5.4(c)), Shapley's game (see Figure 5.4(d)), staghunt (see Figure 5.4(e)), tricky game (see Figure 5.4(f)) [8], and the MASD ($n = 3$, $k = 0.6$) for various values of M ($M + 1 = |A_i|$ for all i) [71].

In comparing the algorithms, we will plot the mean payoffs received by the agents over time. Also of interested is the distribution of payoffs produced by the strategies which the agents learn. We show these results by box and whisker plots, generated by MATLAB. These box and whisker plots show the median (line in middle of box), the median of the upper and lower quartiles (top and bottom of box), and extremes of the data (top and bottom of whiskers). Data points which fall outside of $1.5 * IRQ$, where IRQ is the interquartile range, are considered outliers, and are plotted as $*$'s. Notches in the boxes represent a robust estimate of the uncertainty about the medians for box-to-box comparison. Boxes whose notches do not overlap indicate that the medians of the two groups differ at the 5% significance level. We note that tighter distributions are generally more desirable than distributions which vary widely.

Prisoner's Dilemma. Figure 5.5(a) shows the mean payoffs obtained by the agents over time in the iterated prisoner's dilemma. The mean is obtained from 50 trials (thus, 100 samples since there are two agents). M-Qubed almost always learns to always cooperate, resulting in a payoff of 3 to both agents. The other learners learn strategies that produce inferior payoffs. Note, however, that QL, which stops exploring, is more successful than rQL, which continues exploring. WoLF-PHC generally learns to defect since it plays a stochastic (unpredictable) policy while learning.

The box and whiskers plots for the algorithms in the prisoner's dilemma is shown in Figure 5.5(b). The figure shows that M-Qubed nearly always learns to always cooperate, with the exception of 2 of the 50 trials[8]. In one of the trials the agents learned to alternate between receiving 5 and 0 points per episode. In the other trial in which the agents did not learn mutual cooperation, the cycle $(c, c) - (c, d)$ was played repeatedly. While QL shares the same median payoff with M-Qubed, its payoff distribution varied much more, and was frequently significantly lower than this value. Note that the variance of the distribution of rQL's average payoffs is high

[8]In one trial, M-Qubed converged to alternating between the solutions (C, D) and (D, C), yielding an average payoff of 2.5 to each agent. In the other trial in which M-Qubed did not converge to mutual cooperation, it converged to alternating between the solutions (C, C) and (D, C), yielding an average payoff of 1.5 to one of the agents and 4 to the other.



(a) (b)

Figure 5.5: (a) Average payoffs over time for the iterated prisoners dilemma (in self play). Results are an average of 50 trials. (b) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001–100,000).

due to its continued exploration. Thus, in the iterated prisoner's dilemma, continued exploration can be detrimental to an agent's long-term average payoffs.

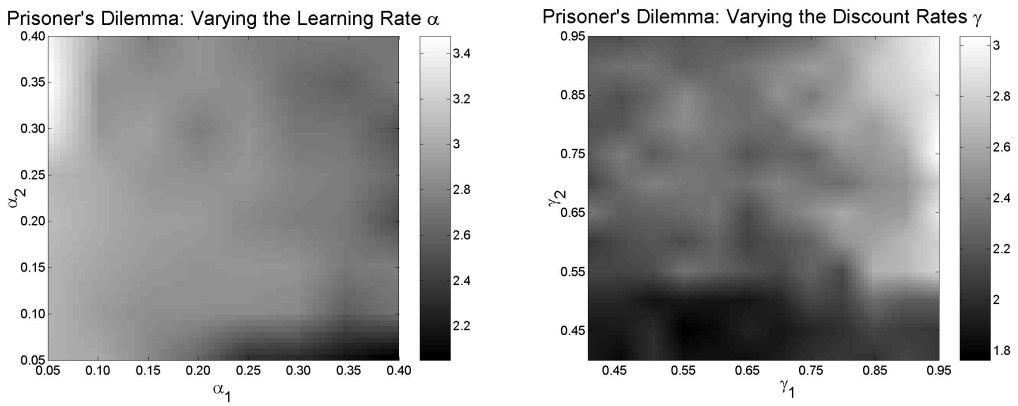
Figures 5.6(a) and (b) show the average payoffs to agent 1 when M-Qubed agents use different values of α and γ_i. The parameter values of the two agents are shown along the x and y-axes respectively. Lighter colors indicate that agent 1 received high payoffs. The figures show that M-Qubed is more successful when using low values of α_i and high values of γ_i. Also, having a slower learning rate (i.e., smaller α_i) than one's associate can sometimes result in a significantly higher average payoff.

Chicken. M-Qubed also learns mutual cooperation in chicken, giving it the highest average payoffs of the learners (see Figure 5.7(a)). Also, M-Qubed's distribution is tighter than the other agents (see Figure 5.7(b)). We note also that QL once again outperforms rQL due to the fact that it stops exploring.

Shapley's Game. Many learning algorithms do not converge in Shapley's game [33, 8]. However, M-Qubed does converge to a cooperative solution which yields an average payoff of $\frac{1}{2}$ to each agent (see Figure 5.8(a)). QL learns to play similarly. The other learners play strategies that yield only slightly lower average payoffs, but do not typically converge.

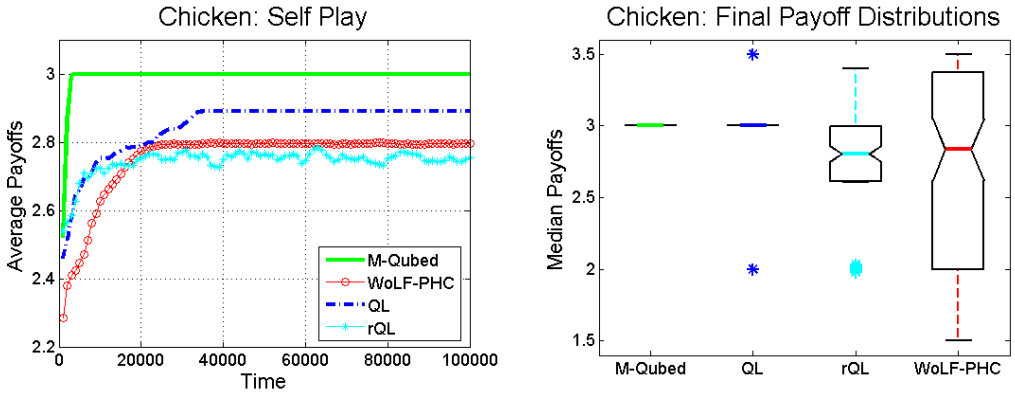
Staghunt. While all the other learners typically learn to play the joing action (d, d) in staghunt, M-Qubed (like the S-Algorithm) learns to play the joint action (c, c), which results in a significantly higher payoff (see Figure 5.9(a)). Figure 5.9(c) shows the results of varying α, which again shows that lower values of α are more successful.

Tricky Game. Figures 5.10(a)-(d) show the performance of both the row and column players in tricky game. As a row player, M-Qubed outperforms the other



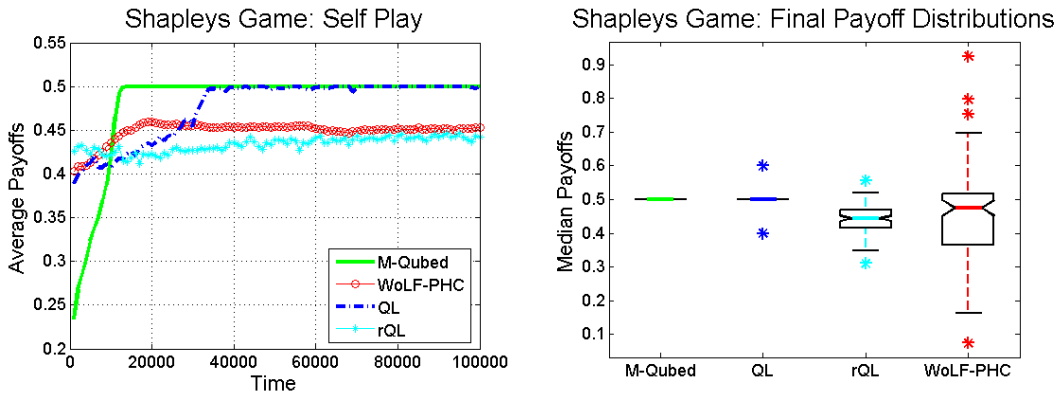
(a) (b)

Figure 5.6: Average payoffs in the iterated prisoner's dilemma to player 1 of learned strategies for M-Qubed in self play when parameter values are varied by the agents as shown along the axes.



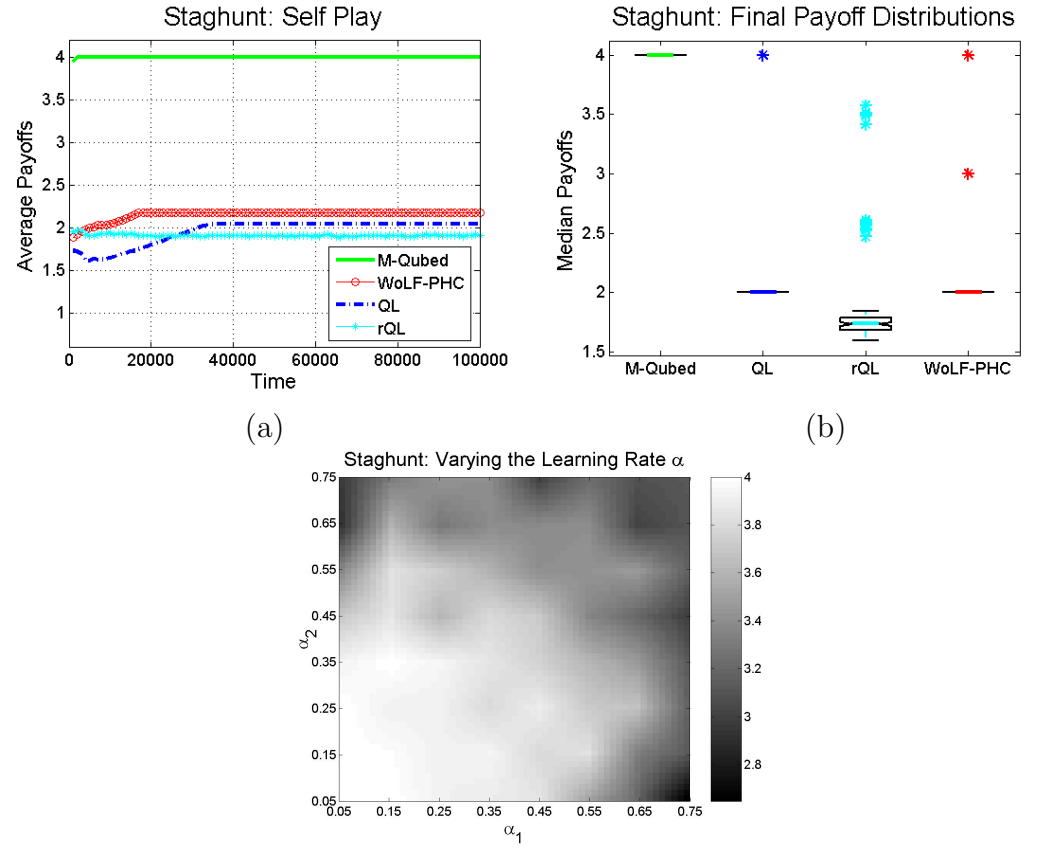
(a) (b)

Figure 5.7: (a) Average payoffs over time for chicken (in self play). Results are an average of 50 trials. (b) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001–100,000).



(a) (b)

Figure 5.8: (a) Average payoffs over time for Shapley's game (in self play). Results are an average of 50 trials. (b) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001–100,000).



(a) (b)

(c)

Figure 5.9: (a) Average payoffs over time for the staghunt game (in self play). Results are an average of 50 trials. (b) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001–100,000). (c) Average payoffs to player 1 when α is varied by the agents.

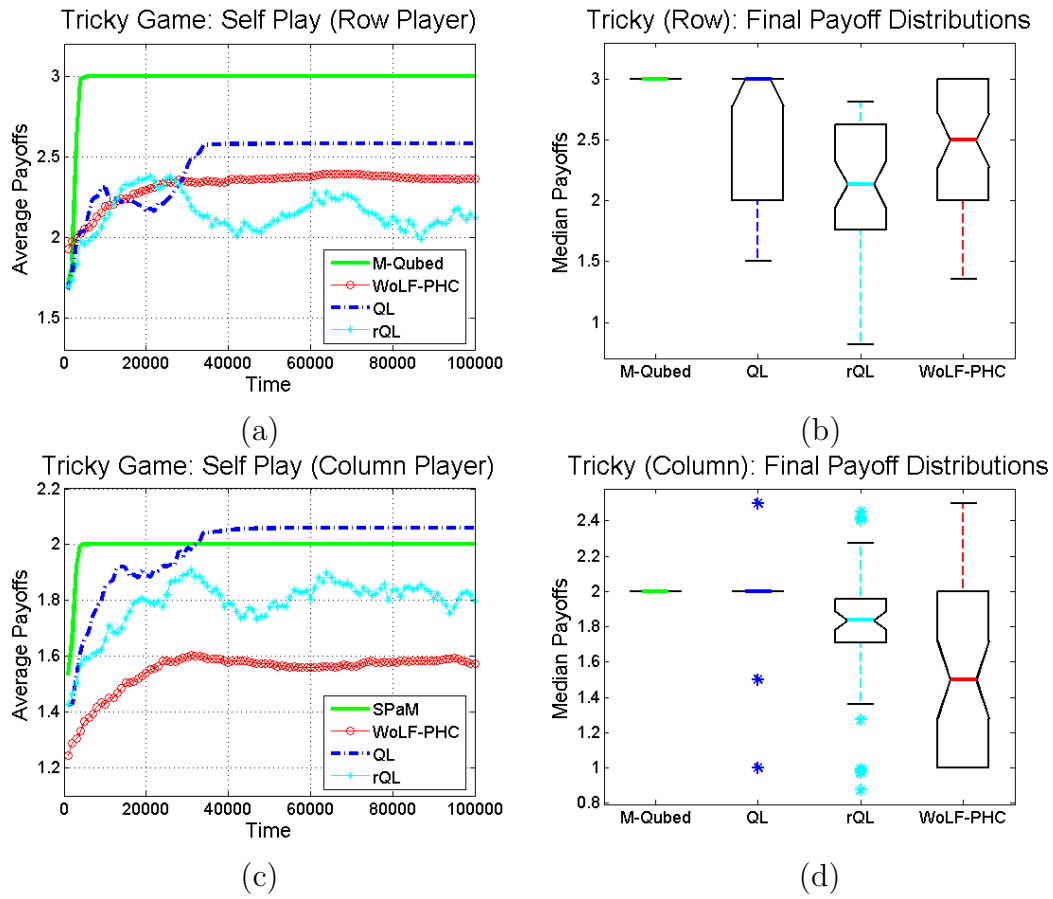
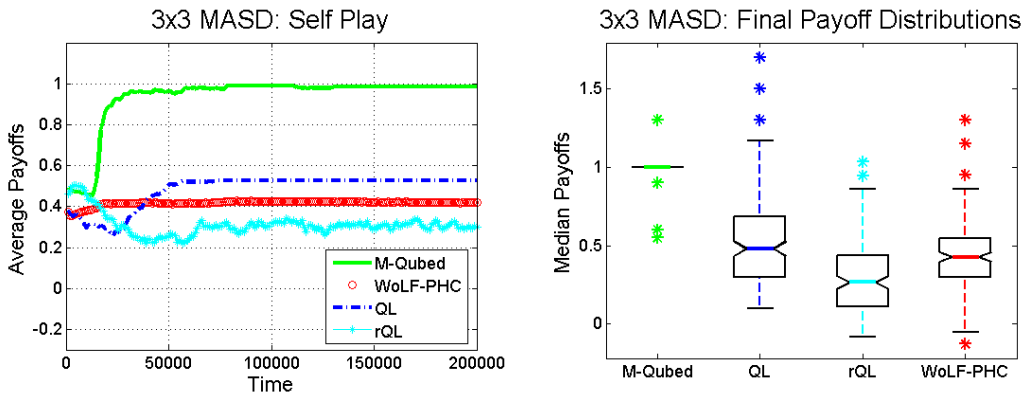


Figure 5.10: (a) Average payoffs over time to the row player in tricky game (in self play). Results are an average of 50 trials. (b) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001–100,000). (c) Average payoffs over time to the column player in tricky game (in self play). Results are an average of 50 trials. (d) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001–100,000).

agents and QL outperforms rQL. However, QL slightly outperforms M-Qubed (in self play) as a column player. This is because QL learns to share high payoffs in a slightly different manner than does M-Qubed. We note, however, that M-Qubed's average payoff (row and column) is much higher than is QL's. Additionally, the variance on learned payoffs is much smaller for M-Qubed.

MASD. We now move to the MASD, described in [71] and reviewed in Section 3.3.2. In this 3×3 game (results shown in Figure 5.11), M-Qubed learns (as does the S-Algorithm) mutual cooperation (i.e., each agent gives all to the group) in almost all trials, while the performance of the other algorithms is much worse (on average).

Figure 5.12 shows the average payoffs to M-Qubed (after learning) in MASDs of 3 to 6 actions. As can be seen, M-Qubed scales quite well in this game, (although



(a) (b)

Figure 5.11: (a) Average payoffs over time in self play of the MASD ($m = 2$, $n = 3$, $k = 0.6$). Results are an average of 50 trials. (b) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 199,001–200,000).

performance does trail off slightly in games with more actions). Note that M-Qubed does not need to vary its learning rates to achieve these results. However, the state space does grow quickly with increasing numbers of actions.

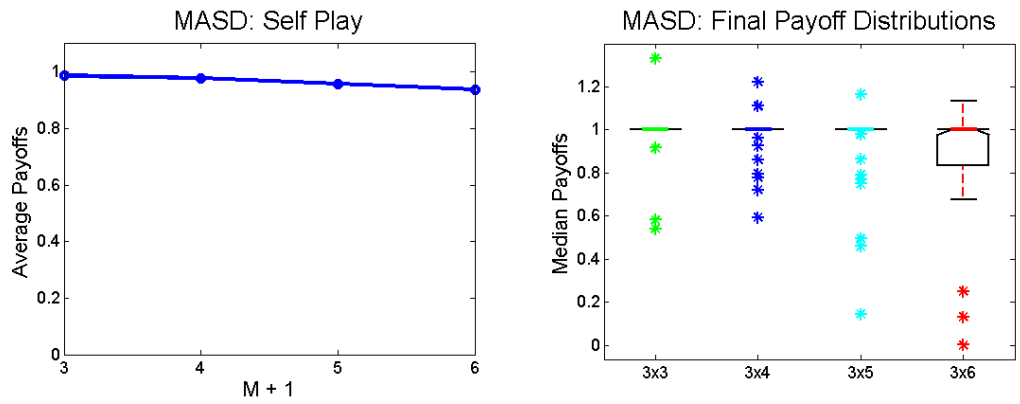
Varying ω_i

In all of the previous games we have analyzed, M-Qubed learns to play cooperatively (in self play), which leads to higher average payoffs (to all agents) over time. We now present a game that M-Qubed (as well as the other learners) have a more difficult time learning. This game is shown in Figure 5.13(left). In this 2×2 game, each agent can guarantee itself no more than a payoff of 0 (its minimax value). However, both agents can receive an average payoff of 0.5 if they alternate between the solutions (C, d) and (D, c).

None of the agents learn to play this game effectively (see Figure 5.13)[9]. However, once again, M-Qubed outperforms the other agents (on average). In most runs, one of the M-Qubed agents learns to play the action that can give it a payoff of 1, while the other agent learns to play randomly, resulting in an average payoff of 0.5 (approximately) to one agent and an average payoff of 0 to the other agent. This can hardly be consider a compromise.

However, M-Qubed is able to learn to offer and accept the profitable compromise of alternating between *winning* and *losing* if it uses a larger history of joint actions (i.e., $\omega_i > 1$). Figure 5.14 shows that both agents receive an average payoff close to 0.5 (in most cases, but not all) when $\omega_1 = \omega_2 = 2$ and $\omega_1 = \omega_2 = 3$.

[9]The performance of rQL is not shown as its behavior is similar to that of QL and WoLF-PHC in this game.



(a) (b)

Figure 5.12: (a) Plot showing the average payoffs of learned strategies (over 50 trials) in the MASD ($n = 3$, $k = 0.6$) for various values of M. (b) Box and whiskers plots of learned strategies.

	c	d
C	0, 0	0, 1
D	1, 0	0, 0

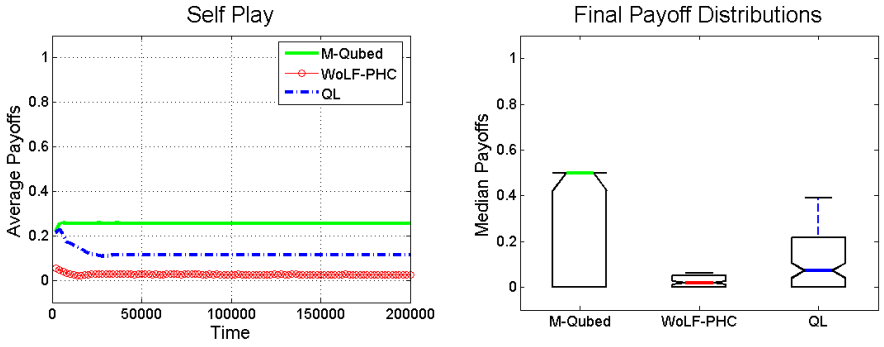


Figure 5.13: Payoffs to the agents in the game shown at left. Although M-Qubed outperforms the other learning algorithms, its play can hardly be considered cooperative (or compromising).

	c	d
C	0, 0	0, 1
D	1, 0	0, 0

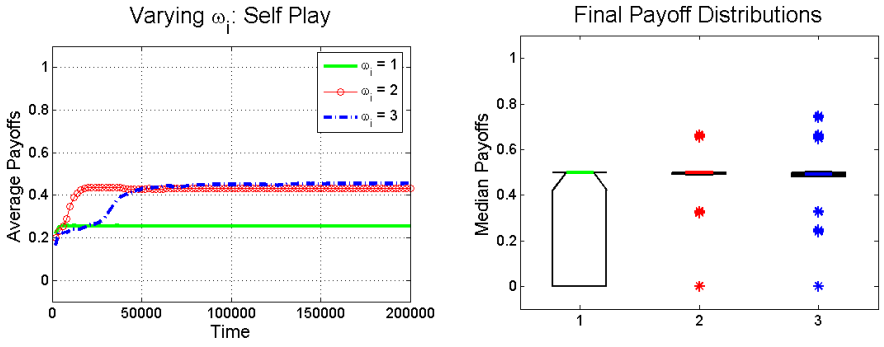


Figure 5.14: Payoffs to M-Qubed in self play for varying values of ω_i in the game shown at left. For $\omega_i > 1$, M-Qubed agents learn a profitable compromise, as the agents learn to alternate between *winning* and *losing*.

We note that learning this cooperative solution (alternating between winning and losing) does not require an agent to consider a history of joint actions greater than 1. However, when $\omega_i = 1$, the Q-values of the agents must become aligned so that the joint action (D, c) is played in the state `Dc` while the joint action (c, D) is played in the state `Cd`. This rarely occurs when $\omega_i = 1$ for each agent i. However, when $\omega_i > 1$, there are multiple opportunities for Q-values to become aligned properly. For example, the joint action (D, c) need only be the solution played by the agents in one of the following states: `Cc-Dc`, `Cd-Dc`, `Dc-Dc`, or `Dd-Dc`.

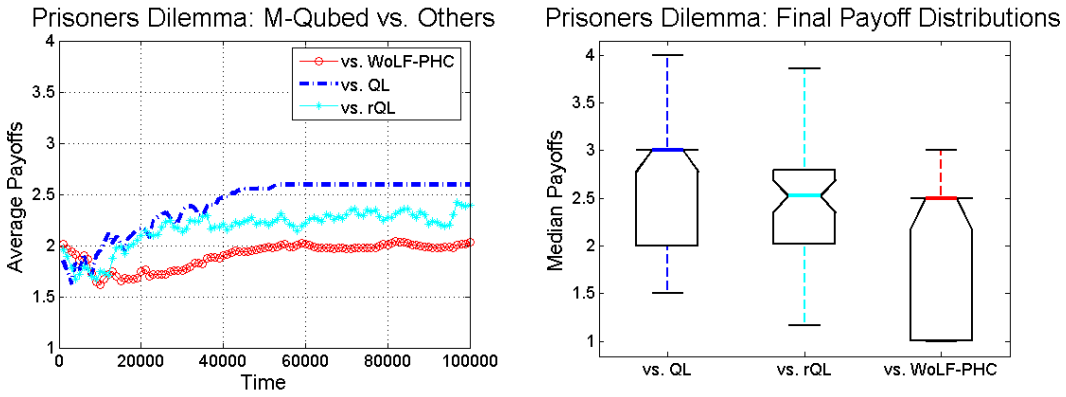
An additional reason for increasing ω_i is to allow M-Qubed to learn when associating with a larger class of agents. For example, when $\omega_1 = 1$, M-Qubed will learn to play optimally when associating with TFT (always cooperates) and Tit-for-2-Tats (alternates between defecting and cooperating), but not Tit-for-3-Tats (TFT3). This is because optimal behavior when associating with TFT3 in the iterated prisoner's dilemma requires that the joint action (c, d) be played twice, followed by the play of the joint action (c, c). Thus, when $\omega_i = 1$, a solution sequence can include no more than one of each solution.

Associating with Other Learners

So far, we have shown that M-Qubed possesses the C/C property in many games in self play. Unfortunately, M-Qubed does not satisfy the C/C property to as great a degree when associating with other learning agents. We show and discuss the performance of M-Qubed when associating with WoLF-PHC, QL, and rQL in two games.

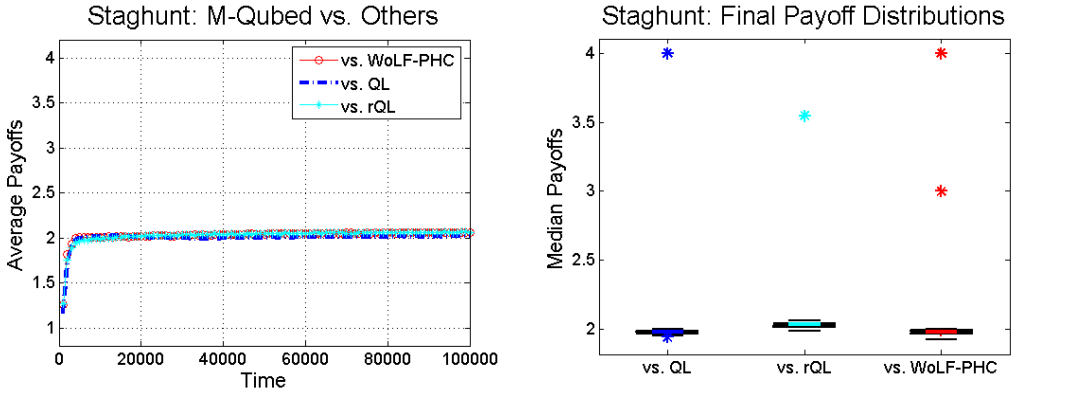
Prisoner's Dilemma. The performance of M-Qubed when it associates with other learners in the prisoner's dilemma is shown in Figure 5.15. As can be seen, M-Qubed receives average payoffs greater than (and never below) the minimax value in most trials when associating with these agents. However, it does not reach mutual cooperation in all trials when associating with any of the agents. We note that M-Qubed does perform better against rQL and WoLF-PHC than these agents do against themselves (see Figure 5.5). Also, M-Qubed performs only slightly worse against QL in this game than QL does against itself.

Staghunt. M-Qubed does not perform as well in the matrix game staghunt as it did in the iterated prisoners dilemma when it associates with other learning agents. Its performance is shown in Figure 5.16. When associating with each agent, M-Qubed learns (as do QL, rQL, and WoLF-PHC in self play; see Figure 5.9) to play its minimax strategy (which is to defect) in nearly all trials with each agent. This result is to be expected somewhat, since it is difficult to teach these algorithms to play cooperatively against such risks, particularly without knowledge of the other agent's payoffs. However, it is possible to teach these algorithms to play cooperatively in this game.



(a) (b)

Figure 5.15: (a) Average payoffs over time to M-Qubed when it associates with other learners in the iterated prisoners dilemma. Results are an average of 50 trials. (b) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001–100,000).



(a) (b)

Figure 5.16: (a) Average payoffs over time M-Qubed when it associates with other learners in the staghunt game. Results are an average of 50 trials. (b) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001–100,000).

5.7 Comparison of M-Qubed and Satisficing Learning

Up to this point in this chapter, we have compared M-Qubed to Q-learning. While M-Qubed is closely related to Q-learning, it is also similar in nature to aspiration-based satisficing learning (as discussed in the previous two chapters).

Like the S-Algorithm, M-Qubed uses a relaxation search to find a profitable strategy. However, while the S-Algorithm relaxes a single variable (its aspirations) until a suitable (mutually satisficing) solution is found, M-Qubed relaxes multiple variables (one for each state-action pair $(h(\omega_i), a_i)$, or Q-value). The relaxation of each Q-value is carried out at a different rate dependent on a) the rewards received, b) the values of subsequent states and c) the frequency with which a state-action pair is visited/updated. This creates a gradient over the Q-values (see Figure 5.2) which can be used to perform informed trembles (i.e., acting randomly with probability η_i when a state with a near maximum global Q-value has not been visited recently).

Like the S-Algorithm, M-Qubed ceases to explore its environment when it becomes satisfied. M-Qubed is satisfied when its current strategy allows it to periodically revisit a state with a near-global Q-value. Viewed in this light, M-Qubed is essentially a satisficing learner.

While M-Qubed and satisficing learning have much in common, M-Qubed has several advantages over the S-Algorithm since it can observe the actions of other agents. First, M-Qubed possesses the security property. Second, M-Qubed can learn when the payoff matrix has non-deterministic payoff functions. Third, M-Qubed is able to learn to alternate between winning and losing in order to reach a profitable compromise. This is made manifest in Shapley's game (see Figures 3.21 and 5.8) and in the results shown in Figure 5.14.

A fourth advantage of M-Qubed over the S-Algorithm is that M-Qubed is able to overcome security and preclusion critical events that exclude cooperative behavior. For example, in the game shown in Figure 5.17 (a game with a security critical event), M-Qubed usually receives payoffs of 4 (row player and 3 (column player), whereas the S-Algorithm does not usually perform so well (see Figure 3.24)[10]. Although this particular game can be solved when an agent learns to tremble its aspirations (see Figure 4.9), this is not always the case in some games with preclusion critical events.

Fifth, M-Qubed can learn a best response when associating with static (stochastic) strategies. For example, in the iterated prisoner's dilemma, the S-Algorithm cannot learn to play a best response to an agent that plays c with probability 0.7 and d with probability c, while M-Qubed does (see Figure 5.18).

[10]Note in Figure 5.17 that WoLF-PHC, QL, and rQL are even more subject to the security critical event than is the S-Algorithm.

	a	b
A	$3.5, 2$	$3.5, 1$
B	$1, 4$	$4, 3$

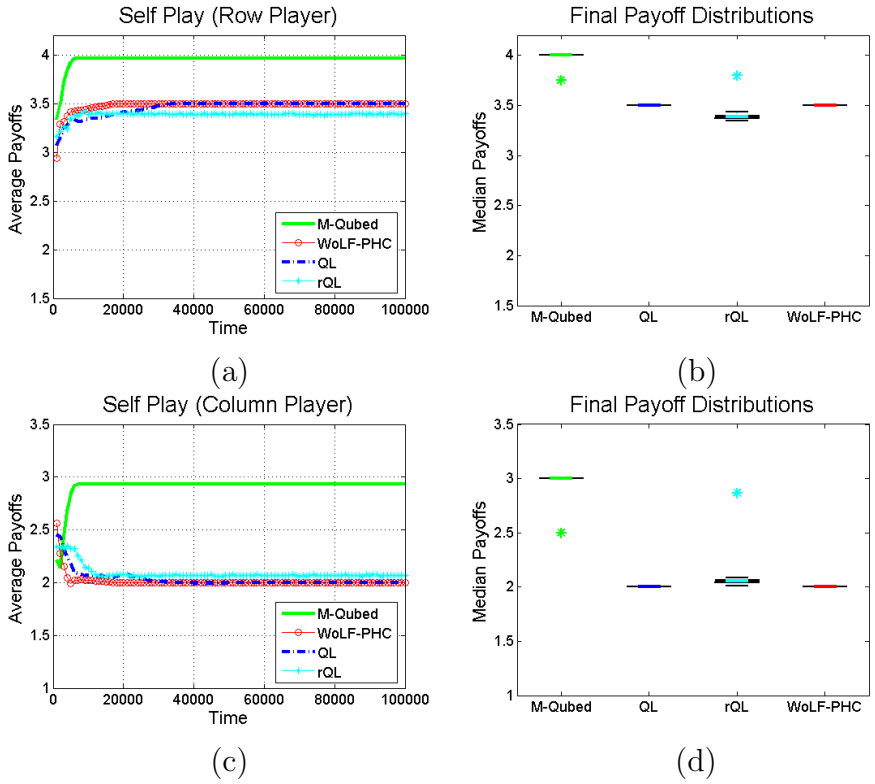


Figure 5.17: (a) Average payoffs over time to the row player in the matrix game shown above (in self play). Results are an average of 50 trials. (b) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001–100,000). (c) Average payoffs over time to the column player. Results are an average of 50 trials. (d) Box and whiskers plots of the final 1,000 episodes (i.e., episodes 99,001-100,000).

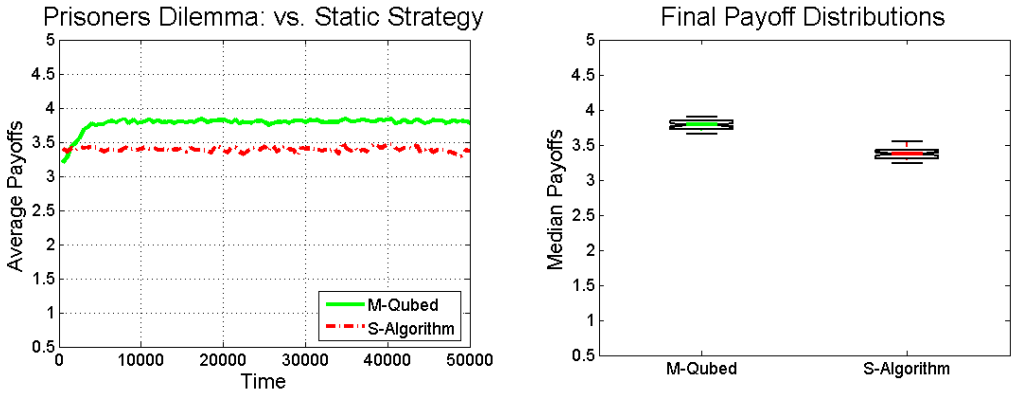


Figure 5.18: Average payoffs over time in iterated prisoners dilemma against a static agent that plays c with probability 0.7 and d with probability 0.3. Results are an average of 50 trials.

5.8 Summary and Discussion

We have advocated that learners in repeated games should a) never learn to receive payoffs below the minimax value of the game and b) have the ability to cooperate/compromise. No algorithm from the literature is known to satisfy both of these properties simultaneously without knowledge of the payoffs of other agents. We presented an algorithm (M-Qubed) that provably satisfies the security property and displays the C/C property (in self play) in many games. In these games, M-Qubed learns (without the knowledge of the payoffs of associates) the solutions proposed in [53], which are pareto efficient. These results support the hypothesis that learning agents should act predictably in non-competitive games, and unpredictably otherwise.

M-Qubed has a number of weaknesses, one of which is that it can be "bullied" (see, for example, [54]) by certain agents (though it still gets a payoff as great as its minimax value)[11]. Additionally, M-Qubed does not have the C/C property when associating with many other learners in many games. We believe that future learning algorithms should use principles such as reputation, teaching, etc. in order to satisfy the C/C property when associating with a larger class of learners [22, 62].

Another weakness of M-Qubed is that it may not be suited for environments in which the intended actions of the agents are not guaranteed to be carried out. Under such circumstances, cooperative behavior is more difficult to achieve since the signals sent by agents (via actions) are not as clear. We note that extending these algorithms to perform well in such environments may require explicit communication (such as the technique used in [67]).

[11] We note, however, that this "weakness" is present (at best) in all other learning algorithms in the literature that do not use information about associates' payoffs.

Chapter 6

M-Qubed in Repeated General-Sum Stochastic Games

6.1 Introduction

In the previous chapters, we limited our discussions to general-sum matrix games. Matrix games form the basis of simultaneous move games. However, they cannot capture "real world" situations in which agents often make and carry out decisions over a sequence of actions and observations. In this chapter, we move to the more general case of stochastic games (also called Markov games). These games are more descriptive than matrix games since they can be designed so that agents can (and must) make and carry out decisions over a sequence of actions and observations.

Learning in repeated stochastic games is more difficult than in matrix games for a number of reasons, of which we mention two. First, the state space of stochastic games is much larger than that of matrix games. Second, the payoffs of the agents are dependent not only on the strategies used by the agents in the current state, but also on the strategies employed by the agents in subsequent states of the episode.

In this chapter, we extend the M-Qubed algorithm that we presented in the previous chapter to infinitely repeated stochastic games. We show (empirically) that the extended M-Qubed algorithm satisfies the same two properties as it does in infinitely repeated matrix games in several important infinitely repeated stochastic games. Multiagent learning algorithms in the literature do not satisfy these two properties in many of these games.

We note that the extended M-Qubed algorithm uses several additional free parameters. Each one of these parameters is added to the algorithm to respond to a different challenge that arises in stochastic games. While we consider this algorithm a first of its kind for learning non-myopic strategies in these games, we do not present a detailed analysis of how changes in the values assigned to these free parameters affect M-Qubed's behavior. Rather, we show that M-Qubed demonstrates the security and compromise/cooperate properties for a set of these parameters.

Given the number of free parameters and the complexity of the algorithm, it is especially important to show that the algorithm is more than an *ad hoc* collection of various rules. In addition to empirically showing that the algorithm possesses the security and compromise/cooperate properties in self-play in many games when certain conditions are met, we also establish a *consistency* property which states that

the extended algorithm does not change the behavior of M-Qubed (as is presented in the previous chapter) in matrix games. Furthermore, we show that, in self play, its performance is comparable or superior to the WoLF-PHC algorithm on a set of interesting infinitely repeated stochastic games.

The structure of this chapter is as follows. In Section 6.2, we formally define stochastic games. Related to stochastic games is some important related work that we have not discussed up to this point. We review some of this work in Section 6.3. In Section 6.4, we overview how M-Qubed can be extended to learn in stochastic games. We present the details of the algorithm in Section 6.5. In Section 6.6, we show the performance of M-Qubed in several important stochastic games. In the final section, we summarize the chapter and discuss future work.

6.2 Stochastic Games

As in the previous chapters, let I be the set of n agents playing the game. Also, let Σ be the set of states (or stages) of the stochastic game. A *stage* is a particular configuration of physical state that the game can take on. This physical state includes the status of matter (walls, gates, etc.) as well as the status (location, velocity, etc.) of the agents playing the game. Let B be the set of possible start (beginning) stages of each episode (which is usually a singleton) of the game, and let G be the set of goal (or terminating) stages of the game. The set of actions available to agent $i \in I$ in stage $\sigma \in \Sigma$ is denoted by $A_i(\sigma)$. Then, let $A(\sigma) = A_1(\sigma) \times \cdots \times A_n(\sigma)$ be the set of joint actions that can be played in stage σ. Let $a_i^t(\sigma) \in A_i(\sigma)$ be the action taken by agent i in stage σ in episode t. Hence, $\mathbf{a}^t(\sigma) = (a_1^t(\sigma), \cdots, a_n^t(\sigma))$ is the joint action taken by the $|I|$ agents in stage σ in episode t.

Let $\sigma_\tau^t \in \Sigma$ denote the τ^{th} stage of episode t. When a joint action is taken in stage σ_τ^t two things happen. First, each agent receives an *immediate reward* (which we denote $\nu_i(\sigma_\tau^t)$ for agent i). $\nu_i(\sigma_\tau^t)$ is a sample taken from the random variable $\mathcal{V}_i(\sigma_\tau^t, \mathbf{a}^t(\sigma_\tau^t))$. Second, when a joint action is taken in stage σ_τ^t, play transitions to the next stage of the episode (denoted $\sigma_{\tau+1}^t$). This transition is governed by the probability transition function $P(\sigma_\tau | \sigma_{\tau-1}, \mathbf{a}^t)$. Once a terminal stage is reached (i.e., $\sigma_\tau^t \in G$), the agents are automatically transitioned to a beginning state (i.e., some stage in B), and a new episode begins. We assume that $\mathcal{V}(\sigma, \mathbf{a})$ and $P(\sigma)$ are stationary for all $\sigma \in \Sigma$ and $\mathbf{a} \in A(\sigma)$.

We are now ready to formally define a stochastic game.

Definition 6.1. *The tuple* $(I, \Sigma, B, G, (A(\sigma))_{\sigma\in\Sigma}, (\mathcal{V}(\sigma, \mathbf{a}))_{\sigma\in\Sigma, \mathbf{a}\in A(\sigma)}, (P(\sigma))_{\sigma\in\Sigma})$ *is a stochastic game, where* I, Σ, B, G, $A(\sigma)$, $\mathcal{V}(\sigma, \mathbf{a})$, *and* $P(\sigma)$ *are defined previously.*

Note that, according to this definition, a matrix game is a special kind of stochastic game. Thus, stochastic games generalize matrix games.

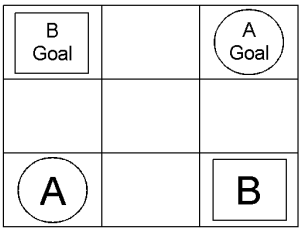


Figure 6.1: Example stochastic game.

As in the previous chapter, we assume that an agent can observe the actions of all agents as well as its own payoffs $\nu_i^t(\sigma)$. Additionally, we assume that agents can observe the current stage (or state) $\sigma \in \Sigma$.

To better understand the terminology given, consider the stochastic game (a coordination game) depicted in Figure 6.1, which is described by Hu and Wellman in [42]. In this game, the two agents begin each episode as indicated in the figure. In each stage, the players simultaneously move in one of the four compass directions. The first agent to reach its goal receives a payoff of 100. In the event of a tie, both agents are rewarded a payoff of 100. Hitting a wall results in a no-op (the player stays put) and if the two players collide (i.e., both try to enter the same position), then they both receive an immediate payoff of -1 and no motion occurs.

Let $(1, 1)$ denote the grid position corresponding to the bottom-left hand corner of the grid (player A's start position), and let the point $(3, 3)$ denote the the upper-right hand corner of the grid. The 4-tuple (x_A, y_A, x_B, y_B) denotes a stage of the game, where (x_A, y_A) and (x_B, y_B) are the players positions on the board. Then, in this game, $\Sigma = \{\{1, 2, 3\} \times \{1, 2, 3\} \times \{1, 2, 3\} \times \{1, 2, 3\}\}$, $B = \{(1, 1, 3, 1)\}$, and $G = \{(1, 3, i, j), (i, j, 3, 1)\}$, where i and j can take on the values 1, 2, and 3.

6.3 Related Work

In this section, we review some work related to learning in stochastic games. First, we briefly review some important elements of reinforcement learning related to estimating future rewards. Second we review work on learning in stochastic games.

6.3.1 Reinforcement Learning

In matrix games (as we discussed in the previous chapter), an agent is faced with learning/predicting the sum of future rewards for taking a particular action in the current episode. To do so, M-Qubed combines the payoff received in the current episode with the estimated value of the subsequent state (which is based on the history of joint actions). In this way, M-Qubed is able to learn/predict the rewards to be received in future episodes.

In repeated stochastic games, an agent must not only predict the sum of rewards to be received in future episodes, but it must also predict the sum of future payoffs to be received within the current episode. To help us solve this problem, we will turn, once again, to principles of standard reinforcement learning.

There are three elementary solution methods of reinforcement learning. These methods are: dynamic programming, Monte Carlo methods, and temporal differencing. The goal of each of these methods is the same: to learn/predict the sum of future rewards resulting from taking a given action from a given state (or stage). However, each of the three methods is suited for different situations. Since dynamic programming requires a model of the state transitions probabilities, and since we are restricting attention to games where the transitions probabilities are not known, we turn to the other two methods (Monte Carlo and temporal differencing) for guidance. We now briefly review these two methods. A more detailed description of each can be found in [75].

In Monte Carlo methods, the value of taking an action is updated using the (possibly discounted) sum of rewards received by the agent from the time it takes the action until the end of the current episode. Then, in a Monte Carlo algorithm, $r_i^t(\sigma_\tau^t)$ is given by

$$r_i^t(\sigma_\tau^t) = \sum_{k=\tau}^{l_t} \gamma^{(k-\tau)} \nu_i^t(k) \tag{6.1}$$

where l_t is the number of stages played in episode t and $\gamma \in (0, 1]$ is the discount factor.

Let $V_i^t(\sigma, a)$ be the estimate of the sum of future rewards at episode t when the action a is taken in stage σ. In Monte Carlo methods, this estimate is given by

$$V_i^t(\sigma, a) = \frac{1}{|T|} \sum_{j \in T} r_i^j(\sigma) \tag{6.2}$$

where T is the set of episodes in which the agent took the action a on the first visit to stage σ and $r_i^j(\sigma)$ is given in Equation 6.1. Note that only the first visit of each stage is counted in the update. This is known as a first-visit Monte Carlo method [75].

One disadvantage of Monte Carlo methods is that they require that an episode be completed before updates can be made. In contrast, temporal differencing typically allows updates to be made much sooner. This is because temporal difference learning *bootstraps*. We say that an algorithm bootstraps if it uses estimated values of subsequent states to predict the sum of future rewards. Q-learning is one example of a reinforcement learning algorithm that bootstraps, as it uses Q-values in the subsequent state to update the Q-values in the current state. Q-learning and M-Qubed (as discussed in the previous chapter) are temporal differencing algorithms.

As we mentioned, M-Qubed uses temporal difference learning to learn the future rewards obtained in subsequent episodes. However, as we will explain in the next

sections, M-Qubed uses a Monte Carlo method to estimate payoffs across a particular episode.

6.3.2 Algorithms for Learning in Stochastic Games

Many of the algorithms in the literature that we discussed in previous chapters are designed for stochastic games. Such algorithms include minimaxQ [51], NashQ [41], WoLF-PHC [9], friend-or-foe Q-learning [52], and Correlated Q-learning [39]. However, we note that as these algorithms do not learn profitable solutions in many matrix games, they perform similarly (if not worse) in stochastic games. We note that each of these algorithms do not estimate future rewards across episodes. Rather, they learn based only on payoffs received in the current episode.

6.4 M-Qubed in Stochastic Games: An Overview

The M-Qubed algorithm as described in the previous chapter is not suitable for stochastic games. In this section, we explain why M-Qubed (as described in the previous chapter) does not learn effectively in these games. Additionally, we present an overview of how M-Qubed can be extended to learn effectively in stochastic games. We leave many of the more formal details until the next section.

6.4.1 The Stage Game

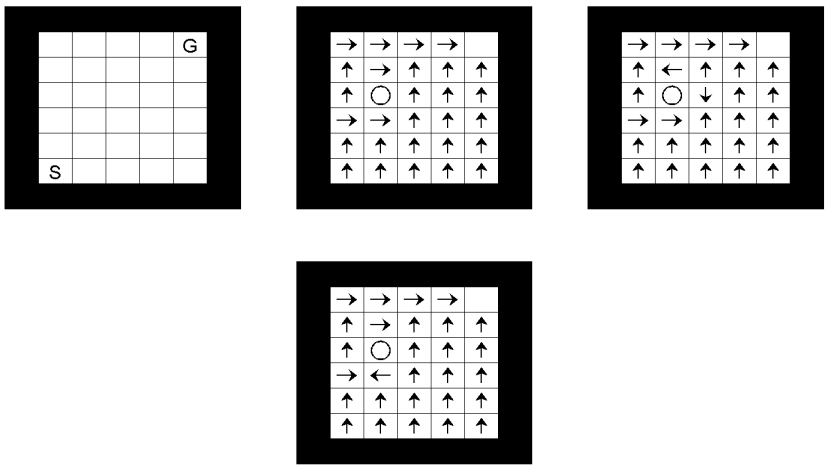
In each stage $\sigma \in \Sigma$ of a stochastic games, the agents play a *stage game*. The stage game is a matrix game defined by the set of players I, the set of joint actions $A(\sigma)$, and the set of payoff matrices $R(\sigma) = \{R_1(\sigma), \cdots, R_n(\sigma)\}$. Let $R_i(\sigma, \mathbf{a})$ be a random variable giving the distribution of agent i's future payoffs when the joint action $\mathbf{a}$ is played in stage σ. The mean of this distribution can be estimated using a temporal differencing method or a Monte Carlo method. We use a Monte Carlo method for reasons that will become clearer in the next section.

Let $r_i^t(\sigma_\tau^t)$ be the sum of future payoffs to agent i in the τ^{th} stage of episode t. Thus, $r_i^t(\sigma_\tau^t)$ is a sample of the random variable $R_i(\sigma_\tau^t, \mathbf{a}^t(\sigma_\tau^t))$. Formally, $r_i^t(\sigma_\tau^t)$ is given by

$$r_i^t(\sigma_\tau^t) = \sum_{j=\tau}^{l_t} \nu_i^t(\sigma_j^t) \tag{6.3}$$

where l_t is the number of stages played in episode t and $\nu_i^t(\sigma_j^t)$ is the immediate payoff to agent i in episode t after the joint action $\mathbf{a}^t(\sigma_j^t)$ was played in stage σ_j^t.

In stochastic games, a separate copy of M-Qubed, one for each $\sigma \in \Sigma$, is used to learn the appropriate mapping from stage to action. Although this extension is obvious and natural, the stage game of a stochastic game has certain qualities that matrix games do not have. In the rest of this section, we discuss these differences and show how M-Qubed can be extended to deal with them.



(a) (b) (c) (d)

Figure 6.2: (a) A simple single-agent grid game. Note how the payoff matrix for the stage $\sigma_{2,4}$ (indicated by the circle) varies according to the agent's strategy in subsequent states. (b)-(d) show example policies.

6.4.2 Comparing the Stage Game with Matrix Games

While the stage game is similar in nature to matrix games, the set of payoff matrices that define the stage game are often non-stationary (though the transition probabilities are stationary). This is because an agent's payoff for the current stage is based on the joint strategies played by the agents in all subsequent stages of the episode. Since the strategies of the agents changes as they learn, the payoffs for a given stage change over time. Thus, the payoff matrices that define the stage game are non-stationary.

To see this, consider the single-agent[1] stochastic game depicted in Figure 6.2(a). In this game, the agent begins each episode in the lower left-hand corner (labeled S) and attempts to move to the cell in the upper right-hand corner (labeled G). From each cell, the agent can move in any of the four compass directions (N, S, E, W). If an action carries the agent into a wall (the black cells) it receives a reward of -1 and remains in the same cell (stage) as it was prior to taking the action. The agent receives 20 points when it reaches the goal. Additionally, the agent incurs a cost (a reward of -1) for each move that it takes. Thus, an agent should learn to reach the goal in as few moves as possible.

To see how changes in the joint strategies used in subsequent states affect $R_i(\sigma)$,

[1]We use a single-agent example for simplicity. However, this discussion applies directly to multiagent scenarios by using joint actions rather than the single agent's actions.

consider the situations depicted in Figure 6.2(b)-(d). In the figures, the agent, located in cell (2,4) (cell (1,1) is the start position S), which is indicated by the circle, must select an action. Let this stage be denoted $\sigma_{2,4}$. Also, let $R_i^t(\sigma_{2,4})$ be agent i's payoff matrix of stage $\sigma_{2,4}$ at episode t, and let $\mu_i^t(\sigma_{2,4}, \mathbf{a})$ (where $\mathbf{a} \in A(\sigma_{2,4})$) be the mean of the distribution $R_i^t(\sigma_{2,4}, \mathbf{a})$.

In the situation depicted in Figure 6.2(b), the agent employs an optimal policy (as indicated by the arrows in each cell) in all (possible) subsequent states (assuming a deterministic transition function). These strategies result in the mean payoffs $\mu_1(\sigma_{2,4}, \mathbf{a})$ (for each $\mathbf{a} \in A(\sigma_{2,4})$) of

N	S	E	W
15	13	15	13

However, if the agent's policies in subsequent states are as indicated in the situation depicted in Figure 6.2(c), then the mean payoffs would be[2]

N	S	E	W
13	≤ 13	≤ 13	13

The distributions of the agent's payoff matrix in stage $\sigma_{2,4}$ would be different still if the agent were to act as indicated in the situation depicted in Figure 6.2(d). In this situation, the means of the entries of the payoff matrix would be

N	S	E	W
15	≤ 13	15	13

Thus, as joint strategies in subsequent states change, the payoff distributions of joint actions that lead to these states can change also. As we mentioned previously, this violates the stationarity assumption of the previous chapter. Thus, M-Qubed will behave differently in the stage game of a stochastic game than it will in a matrix game (with a stationary payoff matrx) unless $R_i(\sigma)$ becomes stationary (or relatively stationary) prior to learning.

As demonstrated in the previous example, the payoff matrix of stage games can be non-stationary in single-agent scenarios as well as in multiagent scenarios. Thus, all reinforcement learning algorithms must deal with this challenge. However, in single-agent scenarios, this is not (theoretically) a problem since each state-action pair must be visited infinitely often (see [75]). Thus, the payoff matrix will eventually become stationary as policies in each stage $\sigma \in \Sigma$ converge[3].

In contrast, many multiagent scenarios require that an agent should act so as to influence the future actions of other agents if it is to learn to play desirable solutions.

[2]For the actions S and E, the policy indicates that the agent would move back and forth forever between two states. We will define rules for breaking these cycles later in this chapter.

[3]Note, however, that if an implementation of a reinforcement learning algorithm does not explore all actions from all states infinitely often, then the non-stationarity of the payoff matrix can become a problem in single-agent domains as well.

M-Qubed attempts to influence the future actions of other agents by ceasing exploration once it becomes *satisfied*[4]. Thus, if M-Qubed becomes satisfied in a particular stage before desirable joint strategies are learned in subsequent stages, then M-Qubed will often learn strategies that produce undesirable payoffs.

6.4.3 Dealing with the Differences

M-Qubed (as described in the previous chapter) cannot learn effectively in stochastic games for two reasons, both of which are caused by the non-stationarity of the payoff matrices of stage games. We now discuss these two reasons and outline how M-Qubed can be extended to overcome these challenges. We omit the formal details until the next section.

The first reason that M-Qubed cannot learn effectively in stochastic games is that $R_i^t(\sigma, \mathbf{a})$ (the random variable that governs the distribution of payoffs to agent i at episode t when the joint action $\mathbf{a} \in A(\sigma)$ is played) is unknown and changing. In the previous chapter, we assumed knowledge of $\mu_i^t(\sigma, \mathbf{a})$ (the mean of the random variable $R_i^t(\sigma, \mathbf{a})$) for each $\mathbf{a} \in A(\sigma)$. However, this assumption is impractical in many stage games.

Being able to estimate $\mu_i^t(\sigma, \mathbf{a})$ for all $\mathbf{a} \in A(\sigma)$ is important for a number of reasons. First, it allows each agent i to compute an estimate of its minimax value $m_i^t(\sigma)$ and minimax strategy $\pi_i^m(\sigma; t)$ at episode t in stage σ. Second, it allows an agent to know $\mu_i^{\max}(\sigma)$ and $\mu_i^{\min}(\sigma)$, which are the maximum and minimum mean payoffs to agent i (at episode t) in stage σ. These values are necessary for setting/updating various parameters, including initializing Q-values. Third, knowing $\mu_i^t(\sigma, \mathbf{a})$ is important for tracking changes (or potential changes) in the agent's payoff matrix. Changes (as well as potential changes) in an agent's payoff matrix can (and should) trigger various behaviors in M-Qubed to help it to overcome myopia in these games.

Thus, extending M-Qubed to learn the stage game of stochastic games requires an estimate of $\mu_i^t(\sigma, \mathbf{a})$ for each $\mathbf{a} \in A(\sigma)$. $\mu_i^t(\sigma, \mathbf{a})$ in stochastic games can be estimated using samples taken from distributions of $R_i(\sigma, \mathbf{a})$ over time. In doing so, M-Qubed must ensure that the environment is sampled sufficiently. Additionally, sampling $R_i(\sigma, \mathbf{a})$ effectively requires a good estimate of *credibility*. Credibility indicates how likely a sample is to be drawn from the final distribution of $R_i(\sigma, \mathbf{a})$ (when strategies have converged in all stages). We describe how M-Qubed does this in Section 6.5.1.

A second challenge for M-Qubed in stochastic games arises from the fact that M-Qubed stops exploring its environment when it becomes satisfied. Thus, care must be taken to ensure that changes in the payoff matrix (or at least important parts of it)

[4]Recall from the last chapter that M-Qubed is satisfied when the strategies of the agents result in visits to some state (history) $s \in S^*$, where S^* is the set of states that have nearly maximum Q-values (for that stage).

	c	d
c	1, 1	0, 5
d	5, 0	1, 1

(a)

	c	d
c	3, 3	0, 5
d	5, 0	1, 1

(b)

Figure 6.3: (a) Possible payoff matrix of the stage game at episode T_0. (b) Possible payoff matrix of the same stage game at episode $T_1 > T_0$. In (b) a more profitable compromise is available, but M-Qubed will not learn it if its relaxation search takes place prior to episode T_1.

do not compromise the relaxation search. The relaxation search can be compromised when an average payoff increases significantly during (or after) the relaxation search.

For example, consider the payoff matrix of a hypothetical stage game shown in Figure 6.3. Let the payoff matrix shown in Figure 6.3(a) give the payoffs of the stage game at episode T_0, and let the payoffs shown in Figure 6.3(b) be the payoffs of the same stage game at episode $T_1 > T_0$ (note that this stage game is a prisoner's dilemma). In the stage game shown at episode T_0, a good compromise (and the one often learned by M-Qubed when $\omega_i > 1$) consists of the agents learning to alternate between the solutions (c, d) and (d, c), which yields an average payoff of 2.5 for each agent. A more profitable compromise can be reached at episode T_1, since the solution (c, c) yields a payoff of 3 to each agent at this time. However, if the agents stop exploring prior to episode T_1 then the agents may not learn this more profitable compromise.

Thus, in order to avoid convergence to undesirable (less profitable) solutions in stochastic games, M-Qubed must be extended so that strategies in (certain) subsequent stages converge before it stops exploring in the current stage. Doing so enables the payoff matrix of a stage to become nearly stationary, or at least stationary in relevant entries; we call this *pseudo-stationarity*. In order to achieve this, we require M-Qubed to restart its learning process in a stage whenever the stage's payoff matrix violates the pseudo-stationarity property. Additionally, M-Qubed in stochastic games estimates the *potential payoffs* of subsequent stages, as these potential payoffs can indicate when the agent should continue to explore (with some probability η) in a given stage. We describe these extensions in greater detail in Section 6.5.2.

6.5 M-Qubed in Stochastic Games: Formal Algorithm

In this section, we formally present extensions to M-Qubed to make it suitable for repeated stochastic games. These extensions take place in two phases. In the first phase (called the sampling phase), M-Qubed estimates its mean payoffs $\mu_i^t(\sigma, \mathbf{a})$. The second phase is the learning phase. In this phase, M-Qubed learns in much the same way as it did in the previous chapter. However, various techniques must be used

in this phase so that M-Qubed does not converge until strategies have converged in (relevant) subsequent stages.

As these discussions require a heavy use of terms that we have already defined (or will define), we provide Table 6.1 as a reference for these terms and their definitions.

6.5.1 Using Samples to Estimate Current Average Payoffs

A sample from the random variable $R_i^t(\sigma, \mathbf{a})$ is obtained in each episode in which the joint action $\mathbf{a}$ is taken in stage $\sigma \in \Sigma$. Denote this sample $r_i^t(\sigma)$. In this subsection, we describe how $\mu_i^t(\sigma, \mathbf{a})$ can be estimated from samples of the distribution $R_i(\sigma, \mathbf{a})$. First, we discuss the *credibility* of a sample. Second, we discuss how samples and credibility can be combined to update $\mu_i^t(\sigma, \mathbf{a})$. Third, we discuss a criteria for sampling $\mu_i^t(\sigma)$ (the mean payoff matrix) sufficiently before learning begins in stage σ. We say that learning has begun in stage σ when Q-values are initialized to $\mu_i^{\max}(\sigma)$.

Credibility

Credibility is an estimate of how likely $r_i^t(\sigma)$ is a sample taken from the final distribution of $R_i(\sigma, \mathbf{a}^t(\sigma))$ after play has converged in all stages. We say that strategies have converged in stage σ if $\boldsymbol{\pi}^t(\sigma, h(\sigma, \omega_i))) = \boldsymbol{\pi}^j(\sigma, h(\sigma, \omega_i))$ for all $j \geq T$ and all histories $h(\sigma, \omega_i)$. This includes exploration. We now describe the way in which M-Qubed estimates the credibility of the sample $r_i^t(\sigma)$.

Let $\kappa_i^t(\sigma)$ be agent i's estimate of the *credibility of the joint strategy* $\boldsymbol{\pi}^t(\sigma)$, which determines the joint action played in stage σ of episode t. Thus, $\kappa_i^t(\sigma)$ is an estimate of the probability that $\boldsymbol{\pi}^t(\sigma)$ will be the joint strategy played by the agents in stage σ when the strategies of all agents have converged in this stage. M-Qubed estimates $\kappa_i^t(\sigma)$ as

$$\kappa_i^t(\sigma) = \begin{cases} 1 & \text{if } \pi_i^t(\sigma) \text{ is } \textit{reliable} \\ \frac{1}{|A_i(\sigma)|} & \text{otherwise} \end{cases} \tag{6.4}$$

where *reliability* is an estimate of whether agent i's strategy $\pi_i^t(\sigma)$ has converged (which we discuss in Section 6.5.3). Thus, when the agent's strategy appears to have converged, then the credibility of the joint strategy is 1. Otherwise, the agent is considered to be exploring, meaning that the joint action has a random chance of being a sample drawn from the final joint strategy. Thus, $\kappa_i^t(\sigma) = \frac{1}{|A_i(\sigma)|}$ in this situation[5].

Notice that Equation (6.4) uses only agent i's strategy $\pi_i^t(\sigma)$ in determining reliability rather than the joint strategy $\boldsymbol{\pi}^t(\sigma)$. This is permissible since M-Qubed's

[5]In Section 6.5.1, we modify $\kappa_i^t(\sigma)$ (slightly) in the event that strategies do not converge in stage σ.

Variable	Meaning
t	The current episode.
l_t	The number of stages played in episode t.
σ_τ^t	The τ^{th} stage of episode t.
$\mathbf{a}^t(\sigma)$	The joint action of the n agents in stage σ of episode t.
$a_i^t(\sigma)$	The action taken by agent i in stage σ of episode t.
$\boldsymbol{\pi}^t(\sigma)$	The joint strategy taken by the agents in stage σ of episode t.
$\nu_i^t(\sigma)$	The *immediate payoff* of agent i in episode t when the joint action $\mathbf{a}^t(\sigma)$ is played in stage σ. Note that $\nu_i^t(\sigma)$ is a sample taken from the distribution $\mathcal{N}_i(\sigma, \mathbf{a}^t(\sigma))$.
$R_i(\sigma)$	Player i's (time generic) payoff matrix for the stage σ.
$R_i(\sigma, \mathbf{a})$	The (time generic) payoff random variable to agent i in stage σ for the joint action $\mathbf{a}$.
$R_i^t(\sigma)$	Player i's payoff matrix on episode t for the stage σ.
$R_i^t(\sigma, \mathbf{a})$	The payoff random variable to agent i on episode t in stage σ for the joint action $\mathbf{a}$.
$r_i^t(\sigma)$	The stage payoff for player i on episode t when the joint action $\mathbf{a}$ is played in stage σ. $r_i^t(\sigma)$ is a sample taken of the random variable $R_i^t(\sigma, \mathbf{a}^t(\sigma))$.
$\mu_i^t(\sigma, \mathbf{a})$	$E[R_i^t(\sigma, \mathbf{a})]$, where $E[\cdot]$ denotes an expected value.
$\mu_i^t(\sigma)$	The payoff matrix with entries $\mu_i^t(\sigma, \mathbf{a})$ for all $\mathbf{a} \in A(\sigma)$.
$m_i^t(\sigma)$	Agent i's estimated minimax value in stage σ at episode t.
$\pi_i^m(\sigma; t)$	Agent i's estimated minimax strategy in stage σ at episode t.
$h^t(\sigma, \omega_i)$	The previous ω_i joint actions played in stage σ prior to episode t.
$\pi_i^t(\sigma, h^t(\sigma, \omega_i))$	Agent i's strategy in stage σ at episode t given the history of joint actions defined by $h^t(\sigma, \omega_i)$.
$\kappa_i^t(\sigma)$	Agent i's estimate of the *credibility of the joint strategy* $\boldsymbol{\pi}^t(\sigma)$
$K_i^t(\sigma)$	Agent i's estimate of the *credibility of the sample* $r_i^t(\sigma)$.
$\Psi_i^t(\sigma, \mathbf{a})$	The accumulated credibility to agent i for the joint action $\mathbf{a}$ in stage σ from episode at which the agent begins to sample its environment through the current episode t.
θ_I	The accumulated credibility needed to sample $\mu_i^t(\sigma, \mathbf{a})$ sufficiently.
$\Omega_i^t(\sigma)$	The set of relevant actions in stage σ at episode t for agent i.
$\zeta_i^t(\sigma)$	Agent i's estimate at episode t of the payoffs it will receive in stage σ when play has converged (in all stages).
$\varepsilon_i^t(\sigma, \mathbf{a})$	The error tolerance for ε-stationarity for $(\sigma, \mathbf{a})$ at episode t.
$t_i^0(\sigma)$	The episode in which agent i last initialized its Q-values (i.e., began learning) in stage σ.
$M_i^t(\sigma)$	Agent i's potential matrix at episode t.
$\Theta^t(\sigma)$	The set of joint actions $\mathbf{a}$ for which $M_i^t(\sigma, \mathbf{a}) > \mu_i^t(\sigma, \mathbf{a}) + \phi$.
$\Theta_i^t(\sigma)$	The set of agent i's actions corresponding to the joint actions in $\Theta^t(\sigma)$.

Table 6.1: Various terms and their definitions.

learned strategies are generally dependent on the strategies of other agents in repeated games. Thus, an agent can assume that as its strategies converge, so will the strategies of other agents. This makes an agent's own strategies an adequate estimate of the reliability of the joint strategy currently being played.

The *credibility of the sample payoff* $r_i^t(\sigma_\tau^t)$ is derived from the credibility of joint strategies used in the subsequent stages of an episode. Let $K_i^t(\sigma_\tau^t)$ be agent i's estimate of the credibility of the sample $r_i^t(\sigma_\tau^t)$. Then, $K_i^t(\sigma_\tau^t)$ is given by

$$K_i^t(\sigma_\tau^t) = \begin{cases} 1 & \text{if } \pi_i^t(\sigma) \text{ is } \textit{reliable} \\ \prod_{j=\tau+1}^{l_t} \kappa_i^t(\sigma_j^t) & \text{otherwise} \end{cases} \tag{6.5}$$

Thus, the credibility of a sample is 1 if the agent considers its strategy to be reliable. Otherwise, it is the product of the credibilities of joint strategies in subsequent stages. Thus, joint actions that lead to stages with converged policies will produce highly credible samples. On the other hand, those joint actions that lead to stages in which policies have not converged will produce less credible samples[6].

In addition to assisting in estimating $\mu_i^t(\sigma, \mathbf{a})$, credibility has a role in updating Q-values and in determining whether a distribution has been sampled sufficiently (see Section 6.5.1).

We are now ready to describe how $\mu_i^t(\sigma, \mathbf{a})$ is estimated.

Using Samples to Estimate Payoffs

After each episode t in which $\sigma \in \Sigma$ is visited, each agent i can update its estimate of the mean payoff $\mu_i^t(\sigma, \mathbf{a}^t(\sigma))$ using

$$\mu_i^{t+1}(\sigma, \mathbf{a}^t(\sigma)) = \left(1 - \lambda_i^\mu(\sigma, t) K_i^t(\sigma)\right) \mu_i^t(\sigma, \mathbf{a}^t(\sigma)) + \lambda_i^\mu(\sigma, t) K_i^t(\sigma) r_i^t(\sigma) \tag{6.6}$$

where $\lambda_i^\mu(\sigma, t) \in (0, 1]$. Thus, the estimated mean is sequentially updated using a convex combination of old estimates and new information.

Sampling Sufficiently Prior to Learning

In the previous chapter, we assumed knowledge of $\mu_i^t(\sigma)$. However, in stochastic games, $\mu_i^t(\sigma)$ is not immediately available. Since M-Qubed must initialize its Q-values to $\frac{\mu_i^{\max}(\sigma)}{1-\gamma_i}$, it must (at the least) know $\mu_i^{\max}(\sigma)$ before it officially begins learning. This is achieved by estimating $\mu_i^t(\sigma)$ during an exploration period prior to beginning learning.

There are two central questions concerning the exploration (or sample) phase. First, how can an agent determine when it has sufficiently explored (i.e., sampled) the environment so that it has an accurate estimate of $\mu_i^{\max}(\sigma)$? Second, how should

[6]We note that [29] attempts a Bayesian approach for estimating the credibility of a sample when learning using Q-learning.

an agent act during this exploration period to ensure that the state space is adequately sampled? We answer these two questions in the following discussion.

First, we define the notion of *accumulating credibility*. Let $T_{t_k}^{t_q}(\sigma, \mathbf{a})$ be the set of episodes in which the joint action $\mathbf{a}$ is taken in stage σ. Then, we say that a joint action $\mathbf{a}$ in state σ *accumulates* X *units of credibility* between episode t_k and t_q (where $t_q > t_k$) if

$$\sum_{j \in T_{t_k}^{t_q}(\sigma, \mathbf{a})} K_i^j(\sigma) \geq X \tag{6.7}$$

where $K_i^j(\sigma)$ is the credibility of the sample obtained in stage σ of episode j.

Let $\Psi_i^t(\sigma, \mathbf{a})$ be the accumulated credibility to agent i for the joint action $\mathbf{a}$ in stage σ from the episode at which the agent begins to sample its environment through the current episode t. Thus, the higher the value of $\Psi_i^t(\sigma, \mathbf{a})$, the more sure the agent becomes of its estimate of $\mu_i^t(\sigma, \mathbf{a})$ (provided that the distribution $R_i(\sigma, \mathbf{a})$ has not changed). M-Qubed begins the learning process (i.e., initializes all of its Q-values to $\frac{\mu_i^{\max}(\sigma)}{1-\gamma_i}$) when $\Psi_i^t(\sigma, \mathbf{a}) \geq \theta_I$ for some $\mathbf{a} \in A(\sigma)$.

The notion of accumulation of credibility allows us to alter (slightly) how M-Qubed estimates the *credibility of the joint strategy* played in the stage σ in episode t ($\kappa_i^t(\sigma)$). When a joint strategy does not converge in a stage for long periods of time, then one might say that the agents' strategies have become "reliably unreliable" in this stage. This suggests that $\kappa_i^t(\sigma)$ should be considered more reliable if play has not converged in this stage while it has accumulated large amounts of credibility. To reflect this, we change $\kappa_i^t(\sigma)$ (from Equation (6.4)) to be

$$\kappa_i^t(\sigma) = \begin{cases} 1 & \text{if } \pi_i^t(\sigma) \text{ is } \textit{reliable} \\ \min\left(1, \frac{1}{|A_i(\sigma)|} + \frac{\Psi_i^t(\sigma, \mathbf{a})}{C}\right) & \text{otherwise} \end{cases} \tag{6.8}$$

where C is a large constant. Note that this equation is identical to Equation (6.4) except that when the joint strategy appears unreliable, the term $\frac{\Psi_i^t(\sigma, \mathbf{a})}{C}$ is added to the random term $\frac{1}{|A_i(\sigma)|}$. If the accumulated credibility $\Psi_i^t(\sigma, \mathbf{a})$ is small, then this change does not affect $\kappa_i^t(\sigma)$ significantly. However, in the event that play does not converge for long periods of time, this assures that $\kappa_i^t(\sigma)$ will eventually go to 1, which is necessary to ensure that the payoff matrix is eventually estimated accurately.

The second question, that of how an agent should act while sampling $R_i(\sigma)$, ties into the trade-off between exploration and exploitation. Since an agent is mostly concerned during this time period with determining $\mu_i^{\max}(\sigma)$, it should (in order to learn quickly) sample the actions that appear to produce the highest payoffs more often than those that appear to produce lower payoffs. At the same time, the agent must be careful to explore all actions sufficiently so that it does not underestimate $\mu_i^{\max}(\sigma)$.

In stochastic games, M-Qubed uses a relaxation search to explore the outcomes of actions in each stage. It does this, as before, by initializing its Q-values to a very high value, and then learning (i.e., updating Q-values) as it would normally. For the duration of this sampling period, M-Qubed's Q-values must remain greater than $\frac{\mu_i^{\max}(\sigma)}{1-\gamma_i}$ to ensure that each stage's payoff matrix is sampled sufficiently. Otherwise, M-Qubed might become satisfied with a solution during the sampling period, which could cause it to cease sampling its environment prematurely.

During the sampling period of a stage (let this stage be σ), Q-values are updated at the end of each episode that σ is visited as follows:

$$Q_i^{t+1}(s^t(\sigma), a_i^t) = (1 - \alpha_i^t(\sigma))Q_t(s^t(\sigma), a_i^t) + \alpha_i^t(\sigma)\left[r_i^t(\sigma) + \gamma_i V_i^t(s^{t+1}(\sigma))\right] \quad (6.9)$$

where $s^t(\sigma)$ is the ordered pair $(\sigma, h^t(\sigma, \omega_i))$ (where $h^t(\sigma, \omega_i)$ is the previous ω_i joint actions taken in stage σ), $\alpha_i^t(\sigma) = \alpha_i K_i^t(\sigma)$, and

$$V_i^t(s^{t+1}(\sigma)) = \sum_{b_i} \pi_i^t(s^{t+1}(\sigma), b_i) Q_i^t(s^{t+1}(\sigma), b_i) \quad (6.10)$$

where $\pi_i^t(s^{t+1}(\sigma), b_i)$ is the strategy that agent i expects to play in the next episode that it visits stage σ given the history $h^{t+1}(\sigma, \omega_i)$. The differences between this Q-update and the one used for matrix games (in the previous chapter) are a) the sample reward $r_i^t(\sigma)$ is used rather than the mean $\mu_i^t(\sigma, \mathbf{a}^t(\sigma))$ and b) the learning rate is multiplied by the sample's credibility.

We note that using the sample reward $r_i^t(\sigma)$ and multiplying the learning rate by the credibility is necessary until $\mu_i^t(\sigma, \mathbf{a}^t(\sigma))$ can be determined with sufficient confidence, at which point M-Qubed updates its Q-values using

$$Q_i^{t+1}(s^t(\sigma), a_i^t) = (1 - \alpha_i)Q_t(s^t(\sigma), a_i^t) + \alpha_i\left[\mu_i^t(\sigma, \mathbf{a}^t(\sigma)) + \gamma_i V_i^t(s^{t+1}(\sigma))\right] \quad (6.11)$$

which is identical to the Q-update used in the previous chapter. We begin using this Q-update equation (rather than the one given in Equation (6.9) when the state-joint action pair $(\sigma, \mathbf{a}^t(\sigma))$ has accumulated θ_I units of credibility. Thus, in essence, Equation (6.9) is the Q-update for the sampling phase and Equation (6.11) is the Q-update for the learning phase.

Two rules must be followed to ensure that the relaxation search during the exploration period adequately samples the agent's environment. They are as follows:

- During the sampling period, $\omega_i = 0$. This causes the agent to explore the potential of each action more thoroughly before it accumulates θ_I units of credibility.
- Only the first visit to a stage in an episode should result in parameter updates. This means that, in subsequent visits to a stage within an episode, the "optimal" policy (i.e., the action with the highest Q-value) need not be used. Rather, if the policy being used in a stage is not reliable, the agent should use a random policy on subsequent visits so that the episode will eventually terminate.

6.5.2 Avoiding Premature Convergence

The second challenge of extending M-Qubed to stochastic games is ensuring (in each stage of the game) that M-Qubed does not converge prematurely. That is, M-Qubed must not converge in any stage until (relevant) subsequent stages have converged. Otherwise, M-Qubed is likely to converge to undesirable strategies since it stops exploring when it becomes satisfied.

When the agent's strategies have converged in (relevant) subsequent stages of stage σ, then relevant entries of the payoff matrix of stage σ will be stationary. We call this kind of payoff matrix *pseudo-stationary*, a term which we will define formally in Section 6.5.2.

In addition to the extension presented in the previous subsection, we extend M-Qubed in two other ways so that strategies in (relevant) subsequent stages converge before the strategy in the current stage. First, we modify M-Qubed so that it restarts the learning process in a stage whenever that stage's payoff matrix violates the property of pseudo-stationarity. We describe this extension in Section 6.5.2. Second, we extend M-Qubed to estimate the *potential payoffs* of subsequent stages, as these estimates of potential payoffs can indicate when the agent should continue to explore (with some probability) in a given stage. We discuss this extension in Section 6.5.2.

Pseudo-Stationarity

In this chapter, we have discussed the *stationarity* of payoff matrices. Formally, we consider the payoff matrix $R_i(\sigma)$ to be *stationary* if, for all $\mathbf{a} \in A(\sigma)$ and for all $j < t$, $\mu_i^j(\sigma, \mathbf{a}) - \mu_i^t(\sigma, \mathbf{a}) = 0$. We say that $R_i(\sigma)$ is *ε-stationary* if, for all $\mathbf{a} \in A(\sigma)$ and for all $j < t$, $|\mu_i^j(\sigma, \mathbf{a}) - \mu_i^t(\sigma, \mathbf{a})| \leq \varepsilon$ for some $\varepsilon > 0$. In general, we say that $R_i(\sigma)$ is stationary if it is ε-stationary for some small ε.

Since $R_i(\sigma)$ is often non-stationary in stochastic games until the joint strategies of all (possible) subsequent stages have converged, it is tempting to delay learning in each stage $\sigma \in \Sigma$ until $R_i(\sigma)$ becomes stationary.[7] Unfortunately, this is not possible (for all σ) since the payoffs of neighboring stages depend upon each other's converged strategies.

However, it is possible to ensure that some of the entries of $R_i^t(\sigma)$ are stationary while M-Qubed learns in stage σ. Thus, rather than requiring that $|\mu_i^j(\sigma, \mathbf{a}) - \mu_i^t(\sigma, \mathbf{a})| \leq \varepsilon$ for all $\mathbf{a} \in A(\sigma)$ and $j < t$, we can require that $|\mu_i^j(\sigma, \mathbf{a}) - \mu_i^t(\sigma, \mathbf{a})| \leq \varepsilon$ for the more *relevant* joint actions (for all $j > t$). That is, let $\Omega_i^t(\sigma) \subset A(\sigma)$ be the subset of *relevant* joint actions in stage σ. Then $R_i^t(\sigma)$ is *pseudo-stationary* if, for all $\mathbf{a} \in \Omega_i^t(\sigma)$ and all $t_i^0(\sigma) < j \leq t$ (where $t_i^0(\sigma)$ is the episode at which agent i initialized its Q-values in σ), $|\mu_i^j(\sigma, \mathbf{a}) - \mu_i^t(\sigma, \mathbf{a})| \leq \varepsilon_i^t(\sigma, \mathbf{a})$, where $\varepsilon_i^t(\sigma, \mathbf{a})$ is some positive value (possibly unique to the stage-joint action pair $(\sigma, \mathbf{a})$).

[7] We say that a stage *begins learning* when it initializes its Q-values to $\frac{\mu_i^{\max}(\sigma)}{1-\gamma_i}$.

	c	d
c	1	0
d	5	1

(a)

	c	d
c	3	0
d	5	1

(b)

Figure 6.4: (a) Payoff matrix of agent i of a stage game at episode T_0. (b) Payoff matrix (of agent i) of the same stage game at episode $T_1 > T_0$. If the threshold $\zeta_i^t(\sigma) > 3$ for all $t \leq T_1$, then this payoff matrix is pseudo-stationary.

We must now define *relevance*. Let $\zeta_i^t(\sigma)$ be agent i's estimate (at episode t) of the average reward that it will receive in stage σ when play has converged in all stages. A joint strategy $\mathbf{a}$ is *relevant* if $\mu_i^t(\sigma, \mathbf{a}) > \zeta_i^t(\sigma) + \varepsilon_i^t(\sigma, \mathbf{a})$. Thus, $\Omega_i^t(\sigma)$ is given by

$$\Omega_i^t(\sigma) = \{\mathbf{a} : \mu_i^t(\sigma, \mathbf{a}) > \zeta_i^t(\sigma) + \varepsilon_i^t(\sigma, \mathbf{a})\} \tag{6.12}$$

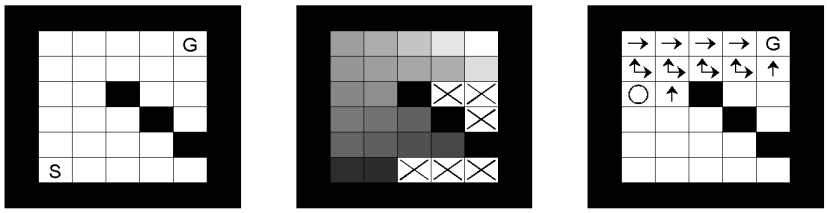
We describe how $\zeta_i^t(\sigma)$ and $\varepsilon_i^t(\sigma, \mathbf{a})$ are determined later in this section.

For an example of pseudo-stationarity, consider Figure 6.4(a), which shows agent i's payoff matrix in a particular stage of episode T_0. Suppose, after episode T_0, that agent i's payoff for the solution (c, c) gradually changes over time until at some episode $T_1 > T_0$, its payoff matrix in this stage is as shown in Figure 6.4(b). If agent i estimates, for all $t \leq T_1$, that its average reward will be greater than $3 - \varepsilon_i^t(\sigma, \mathbf{a})$ when policies have converged in all stages (i.e., $\zeta_i^t(\sigma) > 3 - \varepsilon_i^t(\sigma, \mathbf{a})$ for all $t \leq T_1$), then this payoff matrix is pseudo-stationary (at least through episode T_1).

We note that the definition of relevance just given is not suitable for instances in which an agent plays its minimax strategy $\pi_i^m(\sigma; t)$. When an agent's current strategy is $\pi_i^m(\sigma; t)$, then the agent's minimax value $m_i^t(\sigma)$ is also relevant. Thus, in this circumstance, a payoff matrix is not pseudo-stationary if changes in $R_i(\sigma)$ increase $m_i^t(\sigma)$ by more than $\varepsilon_i^t(\sigma, \mathbf{a})$.

When $R_i(\sigma)$ is pseudo-stationary, M-Qubed is not guaranteed to learn as effectively as it does in matrix games since pseudo-stationarity does not guarantee that strategies have converged in (relevant) subsequent stages. It only indicates that strategies in (relevant) subsequent stages are currently producing (for agent i) samples from a nearly stationary distribution. Thus, if M-Qubed converges prematurely (even if its payoff matrix is currently pseudo-stationary), it can learn behaviors that produce low payoffs. Thus, we will introduce additional exploration methods shortly.

We note that the extensions of M-Qubed described in Section 6.5.1 (the previous subsection) do appear to be sufficient for learning in *deterministic* single-agent games. To see this, consider Figure 6.5(a), which illustrates a simple single-agent maze game. This game has identical physics to the world shown in Figure 6.2. The order in which M-Qubed's strategies converge in each stage in this game is shown in Figure 6.5(b). In the figure, darker colors indicate that convergence occurred at



(a) (b) (c)

Figure 6.5: (a) A simple single-agent grid world game. (b) The order in which the strategy in the stages converged. Darker shading indicates later convergence times. A $\times$ indicates that convergence never occurred in that stage. (c) The converged policies when learning began in stage $\sigma_{1,4}$ (marked by the O). Arrows indicate converged policies. These results were generated using $\theta_I = 40$.

later times. Figure 6.5(c) shows the agent's converged strategies at the episode that Q-values were initialized in stage $\sigma_{1,4}$ (marked by the O). When the agent takes the actions N or E (the agent's optimal policies that it eventually learns) in this stage, the strategies (indicated by the arrows) in all subsequent stages of the current episode have converged. Thus, $R_i(\sigma_{1,4})$ is pseudo-stationary while it is learning.

However, in multiagent environments as well as single-agent environments in which the world is non-deterministic, the extensions of Section 6.5.1 are not sufficient (although they do help).

Restarting Learning

As before, let $t_i^0(\sigma)$ be the episode at which agent i begins learning in stage σ. Then, if $R_i^t(\sigma)$ violates the property of pseudo-stationarity for any $t > t_i^0(\sigma)$, then M-Qubed *restarts the learning process* in stage σ. Restarting the learning process includes re-sampling the environment as discussed in Section 6.5.1, meaning that $\Psi_i^t(\sigma, \mathbf{a})$, agent i's accumulated credibility in stage σ for the joint action $\mathbf{a}$, is set to 0 for each $\mathbf{a} \in A(\sigma)$.

The payoff matrix of a stage game is pseudo-stationary when joint strategies in future stages (at least those reached by relevant[8] joint actions) have converged. Thus, a restart in a stage occurs when joint strategies in these future stages are changing (and, thus, have not converged). Thus, a learning restart allows joint strategies in subsequent stages more time to converge before learning is again attempted in stage σ. The result of this process is that in multiagent and/or nondeterministic environments,

[8]Recall that relevance in stage σ is dependent on the variable $\zeta_i^t(\sigma)$. We discuss how $\zeta_i^t(\sigma)$ is derived later in this section.

as in single-agent deterministic worlds, strategies in stages closer to goals typically converge before strategies in stages further from goals.

A challenge of restarting learning when the payoff matrix of a stage violates the pseudo-stationarity property occurs when (relevant) payoffs in adjacent stages are dependent on each other's strategies. In this case, the two stages might take turns restarting, and, thus, neither would ever converge. Because of this possibility, $\varepsilon_i^t(\sigma, \mathbf{a})$ (which is agent i's error tolerance at episode t for the mean payoff $\mu_i^t(\sigma, \mathbf{a})$) is increased over time based on a) the number of times that agent i has restarted learning in σ up to the current episode t and b) the number of times that it restarted learning when its estimate of converged payoffs was similar to the agent's current estimate of converged payoffs ($\zeta_i^t(\sigma)$). In this way, the requirements of pseudo-stationarity can be relaxed over time (if necessary) until the strategies of the agents in both stages are able to converge.

Formally, let υ_i be agent i's initial tolerance for error given by

$$\upsilon_i = p_i^0 \left(\mu_i^{\max}(\sigma) - \mu_i^{\min}(\sigma)\right) \tag{6.13}$$

where $p_i^0 \in (0, 1)$ is a constant. Let $\mathcal{D}_i(\sigma) = (d_0, d_1, \cdots)$ be the times at which agent i has restarted the learning process in stage σ, where d_j is the episode of the j^{th} restart. Also, let $c_i(\sigma, \zeta_i^t(\sigma))$ be the number of restarts by agent i in stage σ such that $(\mu_i^t(\sigma, \mathbf{a}^t(\sigma) - \zeta_i^{d_j}(\sigma)) < \upsilon|\mathcal{D}_i(\sigma)|$, where $|\mathcal{D}_i(\sigma)|$ is the number of times that agent i has restarted the learning process in stage σ. Then, $\varepsilon_i^t(\sigma, \mathbf{a})$ is given by

$$\varepsilon_i^t(\sigma, \mathbf{a}) = \upsilon B^{c_i(\sigma, \zeta_i^t(\sigma))} \tag{6.14}$$

where $B > 1$ is a constant.

Estimating Potential and Adding Policy Trembles

M-Qubed explores its environment largely by initializing its Q-values high and then relaxing them with experience. Additionally, in matrix games (as we saw in the previous chapter) M-Qubed trembles its policy with probability η_i if it believes that another policy has greater potential than its current policy (see Section 5.5.3).

In stochastic games, M-Qubed trembles its policy with an additional probability η_i^P when it perceives that payoffs obtained by acting in subsequent stages might improve over time. It does this by constructing a *potential matrix* $M_i^t(\sigma)$ for each $\sigma \in \Sigma$ and comparing relevant entries of its potential matrix to the estimated mean payoff matrix $\mu_i^t(\sigma)$. M-Qubed guesses that policies in subsequent states might improve when there exists a joint action $\mathbf{a} \in A(\sigma)$ such that $M_i^t(\sigma, \mathbf{a}) > \mu_i^t(\sigma, \mathbf{a}) + \phi$ (where ϕ is a small positive constant) and $M_i^t(\sigma, \mathbf{a}) > \zeta_i^t(\sigma) + \varepsilon_i^t(\sigma, \mathbf{a})$. In words, M-Qubed guesses that its policies in subsequent states might improve if there is a solution that has a potential payoff higher than its actual estimated payoff, and this solution is

potentially relevant. In such cases, M-Qubed trembles its strategy to play its portion of one such joint action with probability η_i^p.

More formally, let $\Theta^t(\sigma)$ be the set of joint actions $\mathbf{a} \in A(\sigma)$ for which $M_i^t(\sigma, \mathbf{a}) > \mu_i^t(\sigma, \mathbf{a}) + \phi$ and $M_i^t(\sigma, \mathbf{a}) > \zeta_i^t(\sigma) + \varepsilon_i^t(\sigma, \mathbf{a})$. Let $\Theta_i^t(\sigma)$ be the set of player i's actions corresponding to the joint actions in $\Theta^t(\sigma)$. That is, $\Theta_i^t(\sigma)$ is the set of actions $a_i \in A_i(\sigma)$ such that $M_i^t(\sigma, (a_i, \mathbf{a}_{-i})) > \mu_i^t(\sigma, (a_i, \mathbf{a}_{-i})) + \varepsilon$ for some $\mathbf{a}_{-i} \in A_{-i}(\sigma)$. Then, with probability η_i^p, player i plays action $a_i \in \Theta_i^t(\sigma)$ with probability $\frac{1}{|\Theta_i^t(\sigma)|}$, where $|\Theta_i^t(\sigma)|$ is the cardinality of $\Theta_i^t(\sigma)$. Like η_i, η_i^p should be decayed slowly over time.

Because of this additional random exploration, if $M_i^t(\sigma)$ adequately (or optimistically) estimates the potential of subsequent states, its avoids converging prematurely. Thus, it is better for $M_i^t(\sigma, \mathbf{a})$ to overestimate the potential payoffs of subsequent states than to underestimate them, although $M_i^t(\sigma, \mathbf{a})$ should approach $\mu_i^t(\sigma, \mathbf{a})$ (from above) as $t \to \infty$.

The potential matrix $M_i^t(\sigma, \mathbf{a})$ is updated in each episode that $\mathbf{a}$ is played in stage σ using

$$M_i^t(\sigma, \mathbf{a}) = \left(1 - \lambda_i^m(\sigma, t) K_i^t(\sigma)\right) M_i^{t-1}(\sigma, \mathbf{a}) + \lambda_i^m(\sigma, t) K_i^t(\sigma) \left(\nu_i^t(\sigma) + \zeta_i^t(\sigma_{\tau+1}^t)\right) \quad (6.15)$$

where $\lambda_i^m(\sigma, t) \in (0, 1]$ and $\sigma_{\tau+1}^t$ is the subsequent state. Thus, this is a traditional convex combination form of a sequential estimate.

Recall that $\zeta_i^t(\sigma)$ is the value for determining relevance in stage σ. Hence, when $\zeta_i^t(\sigma)$ is optimistic, $M_i^t(\sigma, \mathbf{a})$ will usually be greater than $\mu_i^t(\sigma, \mathbf{a})$ (at least until the joint strategies in subsequent stages converge).

Thus, $\zeta_i^t(\sigma)$ takes on one of two optimistic values, depending on the state of the learning process. If $\max_{\mathbf{a} \in A(\sigma)} \Psi_i^t(\sigma, \mathbf{a}) < \theta_I$, then $\zeta_i^t(\sigma) = \max_{\mathbf{a} \in A(\sigma)} M_i^t(\sigma, \mathbf{a})$. In words, if the stage has not been sufficiently sampled (it is still in the sampling period), then $\zeta_i^t(\sigma)$ is the maximum potential payoff in the stage. However, if the sampling period is completed (and learning has begun), $\zeta_i^t(\sigma)$ is given by $\zeta_i^t(\sigma) = \max_{a_i \in A_i(\sigma), h \in H_{t-T}^{t-1}(\sigma)} Q_i^t(\sigma, h, a_i)$, where $H_{t-T}^{t-1}(\sigma)$ is the set of histories seen in σ in the previous T episodes that σ has been visited. Note that in most cases, $\zeta_i^t(\sigma)$ will be equal to the highest global Q-value in stage σ (if T is big enough). However, this will not always be the case since a given history of joint actions might be unreachable due to the strategies used by other agents.

Formally, $\zeta_i^t(\sigma)$ is given by

$$\zeta_i^t(\sigma) = \begin{cases} \max_{\mathbf{a} \in A(\sigma)} M_i^t(\sigma, \mathbf{a}) & \text{if } \max_{\mathbf{a} \in A(\sigma)} \Psi_i^t(\sigma, \mathbf{a}) < \theta_I \\ \max_{a_i \in A_i(\sigma), h \in H_{t-T}^{t-1}(\sigma)} Q_i^t(\sigma, h, a_i) & \text{otherwise} \end{cases} \quad (6.16)$$

As we discussed previously, after each episode in which $\mathbf{a}$ is played in stage σ_τ^t, the potential payoff $M_i^t(\sigma_\tau^t, \mathbf{a})$ is updated using the potential estimate of the next stage, which is $\zeta_i^t(\sigma_{\tau+1}^t)$. However, when Q-values are re-initialized after learning restarts in

stage σ (after its sample period), $\zeta_i^t(\sigma)$ can change suddenly (usually in the upward direction). As a result, potential payoffs estimated in adjacent stages that were based (in part) on previous values of $\zeta_i^t(\sigma)$ can become pessimistic unless they are corrected.

Hence, we must have a method for correcting pessimistic potential payoff estimates. Let $\mathcal{Z}_i^\sigma(\vartheta, \mathbf{a})$ be the average value of $\zeta_i^t(\sigma)$ used to update the potential payoff $M_i^t(\vartheta, \mathbf{a})$ (where $\vartheta \in \Sigma$) in past episodes and let $\Upsilon_i^\sigma(\vartheta, \mathbf{a})$ be the percentage of $M_i^t(\vartheta, \mathbf{a})$ that has been influenced (by updates using Equation (6.15)) by past estimates of $\zeta_i^t(\sigma)$.[9] Then, when Q-values are re-initialized in stage σ, the following correction is made

$$M_i^t(\vartheta, \mathbf{a}) = M_i^t(\vartheta, \mathbf{a}) + \left(\zeta_i^t(\sigma) - \mathcal{Z}_i^\sigma(\vartheta, \mathbf{a})\right) \Upsilon_i^\sigma(\vartheta, \mathbf{a}) \tag{6.17}$$

Thus, $M_i^t(\vartheta, \mathbf{a})$ is changed to reflect the idea that the stage σ always had a potential of $\zeta_i^t(\sigma)$. Note that this correction must be made for each $(\vartheta, \mathbf{a})$ that the stage σ has influenced.

We have previously used the term *reliability*, which estimates whether or not a strategy has converged in a given stage. We now describe how this estimate is constructed.

6.5.3 Reliability

Reliability estimates whether a strategy has converged in a given stage σ. M-Qubed uses two different indicators to determine the reliability of a stage's current strategy. They are a) how much credibility has accumulated since a strategy for any $h^t(\sigma, \omega_i) \in H^t(\sigma, \omega_i)$ last changed in episode t_c and b) whether or not there exists a joint action $\mathbf{a} \in \Omega_i^t(\sigma)$ such that $M_i^t(\sigma, \mathbf{a}) > \mu_i^t(\sigma, \mathbf{a}) + \varepsilon$ (i.e., whether potential estimates reflect actual payoffs). If the stage has accumulated T_C units of credibility since episode t_c and there does not exist a joint action $\mathbf{a} \in \Omega_i^t(\sigma)$ such that $M_i^t(\sigma, \mathbf{a}) > \mu_i^t(\sigma, \mathbf{a}) + \varepsilon$, then the strategy in that stage is considered reliable.

6.5.4 Algorithm Summary

We have now formal described how M-Qubed is extended to repeated stochastic games. In this subsection, we summarize the algorithm. First, we review the extensions discussed in this chapter. Second, we discuss how these extensions do not affect M-Qubed's behavior is matrix games. Third, we discuss how these extensions affect how an agent selects its actions in the stage game. Fourth, we review the parameters that must be updated (at the end of the episode) in each stage visited in the episode.

[9] Note that both $\mathcal{Z}_i^\sigma(\vartheta, \mathbf{a})$ and $\Upsilon_i^\sigma(\vartheta, \mathbf{a})$ should be updated after each update of $M_i^t(\sigma, \mathbf{a})$.

Extensions Summary

As described in this section, M-Qubed (as described in the previous chapter) must be extended in three ways so that it can learn successfully in the stage game of a stochastic game. We now review these extensions.

First, since an agent's payoff matrix of a stage of a stochastic game is unknown, it must be estimated using sample payoffs obtained from acting in the environment. This can be a difficult task, however, since the payoff matrix can be non-stationary due to its dependence on joint strategies played in subsequent stages of an episode. Because of this, M-Qubed associates a measure of *credibility* to each sample payoff, which is used to determine how the sample payoff should affect the agent's mean payoff estimate. Credibility is also used to determine when the payoff matrix has been sufficiently sampled so that learning can begin in this stage. These extensions to M-Qubed were described in Section 6.5.1.

Second, since the payoff matrix of a stage game is non-stationary, various measures must be taken to ensure that M-Qubed does not stop exploring prematurely. One method used by M-Qubed to avoid premature convergence is to require that its payoff matrix be pseudo-stationary while it learns. A pseudo-stationary matrix has certain (relevant) entries that remain nearly stationary. If the payoff matrix of the stage game violates the property of pseudo-stationarity any time after learning has begun, then M-Qubed restarts the learning process in this stage (which begins by re-sampling the environment). These extensions are given in Section 6.5.2.

The third extension of M-Qubed that we have discussed in this section is also made to ensure that M-Qubed's strategy does not converge prematurely in any stage. This extension involves estimating a potential payoff matrix, which estimates the potential payoffs that joint actions from a given stage will produce when the strategies of subsequent stages converge. If entries of the potential payoff matrix exceed the agents actual estimates (generated by sample payoffs), then M-Qubed explores its environment with additional probability. In this way, M-Qubed can avoid converging prematurely to strategies that produce undesirable payoffs.

Matrix Games Revisited

We now discuss how these extensions affect M-Qubed's behavior in matrix games (from what we saw in the previous chapter), which are a special class of stochastic games.

In the previous chapter, we assumed that the agent knew its payoff matrix prior to learning. However, in this chapter we have assumed that the payoff matrix is unknown. Thus, M-Qubed must estimate its payoff matrix during an exploration period prior to learning. However, if M-Qubed samples its payoff matrix sufficiently, then the other extensions (presented in this chapter) do not change the behavior of M-Qubed in matrix games (as presented in the previous chapter).

To see this, consider each extension made to the algorithm. The first extension which we consider is credibility, which affects how parameters are updated. However, in matrix games, there are no subsequent states, so the credibility of each sample is 1. Thus, credibility does not affect M-Qubed in matrix games.

Second, we consider how restarts affect M-Qubed. M-Qubed restarts the learning process whenever its payoff matrix appears to violate the property of pseudo-stationarity. However, in matrix games, the payoff matrix is stationary, so it is also pseudo-stationary. Thus, if the environment is sampled sufficiently prior to learning, then M-Qubed will never restart its learning process in matrix games. Thus, restarts do not change the behavior of M-Qubed in matrix games.

Lastly, we consider how potential payoff estimates affect the behavior of M-Qubed in matrix games. Since matrix games do not have subsequent states in an episode, the potential payoff estimate will be identical to the actual payoff estimates if the payoff matrix has been sampled sufficiently and if $\lambda_i^{\mu}(\sigma, t) = \lambda_i^{m}(\sigma, t)$ (see Equations (6.6) and (6.15)). Since both $\lambda_i^{\mu}(\sigma, t)$ and $\lambda_i^{m}(\sigma, t)$ should approach zero (or some low value) as time goes on, this requirement is fulfilled. Thus, this extension does not affect M-Qubed's exploration strategies in matrix games.

Hence, this extended version of M-Qubed behaves the same in matrix games as the algorithm presented in the previous chapter.

The extensions presented in this chapter introduce changes in both the action selection and parameter update components of the algorithm. We now review these changes.

Action Selection

M-Qubed determines its strategy in a stage of a stochastic game in the same way as explained in the previous chapter (see Section 5.5.2), with two exceptions. The first difference has to do with the treatment of multiple visits to a stage during an episode. The second difference is the additional exploration introduced by differences between the potential payoff matrix $M_i^t(\sigma)$ and the actual mean payoff matrix $\mu_i^t(\sigma)$.

While an agent is learning in stochastic games, its current strategies do not always result in the agent entering a goal state. Thus, to avoid long cycles in an episode, an agent acts randomly when it visits a stage for a second (or more) time in an episode. The exception to this rule is when the agent believes that its policy in a stage is reliable (see Section 6.5.3), at which time the agent acts according to its current strategy.

When the potential payoff matrix $M_i^t(\sigma)$ and the actual mean payoff matrix $\mu_i^t(\sigma)$ do not agree, M-Qubed explores its environment with higher probability than otherwise. Otherwise, it selects a strategy according to the rules defined in the previous

chapter. Thus, M-Qubed in stochastic games selects its strategy in the following way:

$$\pi_i^t(\sigma, h^t(\sigma, \omega_i)) \rightarrow \begin{cases} \text{rand}(\Theta_i^t(\sigma)) & \text{w/ prob } \eta_i^P \text{ when } \Theta_i^t(\sigma) \neq \emptyset \\ \text{Equation 5.12} & \text{otherwise} \end{cases} \tag{6.18}$$

where $\text{rand}(\Theta_i^t(\sigma))$ is a random selection from the set $\Theta_i^t(\sigma)$ (which is defined formally in Section 6.5.2).

We note one other exception, which is not reflected in Equation (6.18). This exception is strategy selection during the exploration (or sampling) period. During this period, M-Qubed's actions are determined solely by its Q-values (it does not explore its environment). Thus, in these cases, $\pi_i^t(\sigma, h^t(\sigma, 0))$ is given by

$$\pi_i^t(\sigma, h^t(\sigma, 0)) \rightarrow \underset{a_i \in A_i(\sigma)}{\text{argmax}}\, Q_i^t(\sigma, h^t(\sigma, 0), a_i) \tag{6.19}$$

Note the use of $\omega_i = 0$, which is in accordance with our discussion in Section 6.5.1.

Parameter Updates

At the end of each episode t that stage σ is visited, M-Qubed in stage σ updates the following parameter values:

- $M_i^{t+1}(\sigma, \mathbf{a}^t(\sigma))$ using Equations (6.15) and (6.17)
- $\mu_i^{t+1}(\sigma, \mathbf{a}^t(\sigma))$ using Equation (6.6)
- $\zeta_i^{t+1}(\sigma)$ using Equation (6.16)
- $\pi_i^{t+1}(\sigma, h^t(\sigma, \omega_i))$ using Equation (6.18) or Equation (6.19) (depending on whether or not the sample period is complete)
- $\Psi(\sigma, \mathbf{a}^{t+1}(\sigma))$ as discussed in Section 6.5.1
- $Q_i^{t+1}(\sigma, h^t(\sigma, \omega_i), a_i^t(\sigma))$ using Equation (6.11) if $\Psi_i^{t+1}(\sigma, \mathbf{a}^t(\sigma)) \geq \theta_I$ or Equation (6.10) otherwise
- $m_i^{t+1}(\sigma)$
- $\pi_i^m(\sigma; t+1)$
- $\acute{\beta}_i^{t+1}(\sigma)$ and $\beta_i^{t+1}(\sigma)$ as in the previous chapter.

We emphasize that these updates only occur for the first visit to each stage in the episode.

Name	Parameter details
M-Qubed	$\alpha_i = 0.1$, $\gamma_i = 0.95$, $\omega_i = 1$, $\theta_I = 20$, $\eta_i = 0.04\left(\frac{20000}{20000+t}\right)$, $\lambda = \frac{0.01\alpha_i}{\mu_i^{\max}-\mu_i^{\min}}$, $p_i^0 = 0.02$ (see Equation (6.13)), $B = 1.5$ (see Equation (6.14)), $\lambda_i^{\mu}(\sigma,t) = \max\left(0.01, \frac{1}{\Psi(\sigma,\mathbf{a}^t(\sigma))}\right)$ (see Equation (6.6)), $\lambda_i^{m}(\sigma,t) = \max\left(0.01, \frac{1}{\sqrt{\Psi(\sigma,\mathbf{a}^t(\sigma))}}\right)$ (see Equation (6.15))
WoLF-PHC	$\alpha_i = 1/(10 + 0.001\kappa_s^{a_i})$, $\gamma_i = 0.95$, $\delta_w = 1/(1000.0 + \frac{\kappa_s^{a_i}}{10})$, $\delta_l = 4\delta_w$

Table 6.2: Learners and their parameter values. $\kappa_s^{a_i}$ is the number of times that action a_i has been played in state s.

6.6 Results

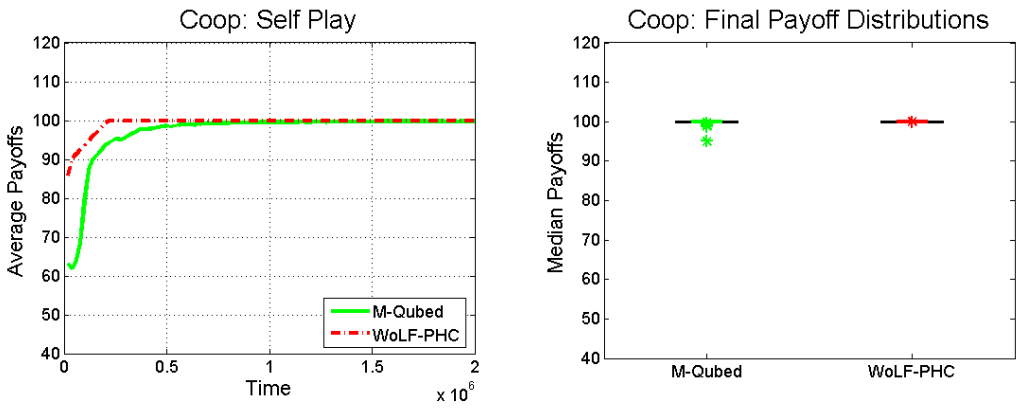
Our extension of M-Qubed to stochastic games requires the use of several additional free parameters. Each one of these parameters has been designed to deal with one or more challenges of learning in stochastic games. In this section, we demonstrate that these parameters can be set so that M-Qubed displays the security and compromise/cooperate properties (at least in self play) in many stochastic games. We note that, while the behavior of the algorithm does not seem to change significantly with small changes in parameter values in most situations, these results might not hold for all sets of parameter values. However, we do not include a complete analysis of how these free variables affect the performance of the algorithm.

In this section, we compare the performance of M-Qubed in self play with the performance of WoLF-PHC in self play in a number of stochastic games. We choose to compare the performance of M-Qubed to WoLF-PHC since WoLF-PHC demonstrates typical behavior of multiagent learning algorithms found in the literature in these games. Unless stated otherwise, all results in this chapter are generated using the parameter values shown in Table 6.2.

A number of stochastic games have been presented and analyzed in recents years. These games have similar characteristics to the matrix games we have analyzed in past chapters. We describe five of these games and compare the performance of M-Qubed and WoLF-PHC in them.

6.6.1 A Coordination Game

First, we discuss the performance of the algorithms in the coordination game that we described in Section 6.2 (see Figure 6.1). In this game, both of the agents can receive the maximum payoff of 100 if they learn to coordinate their actions (to avoid



(a) (b)

Figure 6.6: Average payoffs over time to the agents (in self play) for the coordination game depicted in Figure 6.1.
(a) Average payoffs over time to the agents (in self play) for the coordination game depicted in Figure 6.1. Results are an average of 25 trials. (b) Box and whisker plots showing the distribution of the agents' payoffs after 2000000 episodes.

collisions) while moving toward the goal. Both M-Qubed and WoLF-PHC learn to do so in this game, as shown in Figure 6.6.

6.6.2 Prisoner's Dilemma

Goodrich *et al.* presented a multi-stage (stochastic game) version of the prisoner's dilemma in [36]. We analyze the performance of M-Qubed and WoLF-PHC in a smaller version of this game, which is shown in Figure 6.7. In this game, two agents (labeled `A` and `B`) begin on opposites sides (corners) of the world. The world is divided by a wall containing four different gates which are open to begin each episode. The goal of each agent is to move to the other side of the world, at which point it receives a payoff of 10. However, the agent also loses 1 point for every move it takes, so the agents should try to move to the other side of the world in as few moves as possible. The "physics" of the world are as follows:

- Agents may move up, down, left, and right.
- Moves into walls or closed gates result in the agent remaining where it was before the action was taken.
- If one agent moves through gate 1 and the other agent does not, then gates 1, 2, and 3 close (after the defecting agent moves through the gate). This is demonstrated in the situation shown at the bottom left of Figure 6.7. The result of this action sequence is that the agent that moves through gate 1 receives a

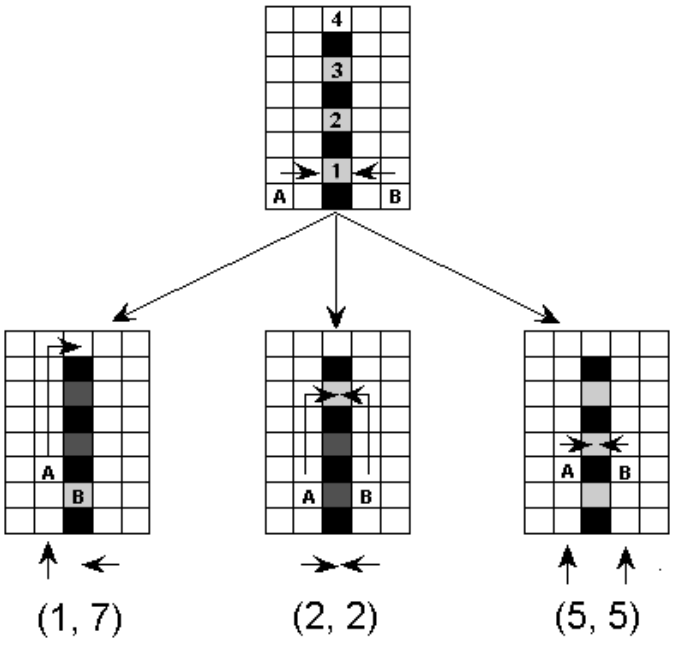
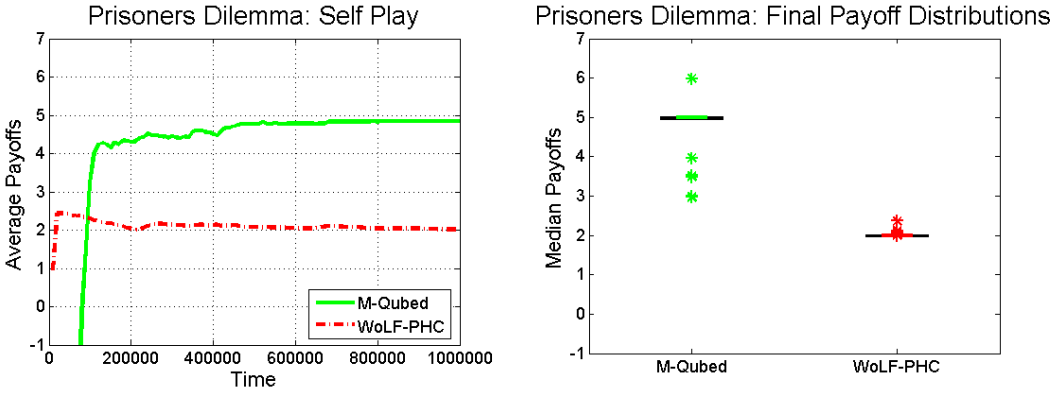


Figure 6.7: Stochastic game version of the prisoner's dilemma.



(a) (b)

Figure 6.8: (a) Average payoffs over time to the agents (in self play) for the prisoner's dilemma game depicted in Figure 6.7. Results are an average of 25 trials. (b) Box and whisker plots showing the distribution of the agents' payoffs after 1000000 episodes.

net payoff of 7, while the other agent receives a net payoff of 1 (assuming the agents do not waste moves).

- If both agents arrive and attempt to move through gate 1 at the same time, gates 1 and 2 close (without allowing either of the agents to pass through the gate). This situation is demonstrated in the situation depicted at the bottom center of Figure 6.7. Both agents receive a net payoff of 2 in this case.

- If both agents move through gates 2, 3, or 4, then none of the gates close (and, thus, neither of the agents are denied passage). The situation in which the agents both move through gate 2 is shown in the bottom right of Figure 6.7 and results in both agents receiving a net payoff of 5.

Note that the net payoffs just mentioned define a prisoner's dilemma. Thus, moving through gate 1 can be considered *defection*, and moving through gate 2 can be considered *cooperation*.

Due to the non-stationarity of the stage game, learning non-myopic solutions (i.e., learning to offer and accept profitable compromises) in this multi-stage prisoner's dilemma is more difficult than learning non-myopic solutions in the equivalent prisoner's dilemma matrix game. Nevertheless, M-Qubed still learns mutual cooperation most of the time, resulting in an net average payoff of nearly 5 to each agent (in each episode). This result is shown in Figure 6.8. This figure shows the results of 25 trials, of which M-Qubed learned mutual cooperation 22 times (resulting in a net average payoff of 5 to both agents). In the first trial that M-Qubed did not learn mutual cooperation, the agents learned to alternate between the solutions (D, D) and (C, C), resulting in an average net payoff of 3.5 to both agents. In the second trial that it did not learn mutual cooperation, the agents learned to alternate between the solutions (D, C) and (C, C), resulting in average net payoffs of 6 and 3, respectively. In the last such trial, the M-Qubed agents learned to alternate between the solutions (D, C) and (C, D), resulting in the average net payoffs of 4 and 3^{10}, respectively. Thus, even in the instances in which M-Qubed did not learn mutual cooperation, it was able to make and accept compromises that yield average net payoffs greater than its minimax value (which is 2). Meanwhile, WoLF-PHC learns mutual defection in self play in all 25 trials, resulting in a net payoff of 2 to each agent.

Note that M-Qubed learns quite slowly in this game. Thus, it receives very low payoffs for nearly the first 100000 episodes of the game.

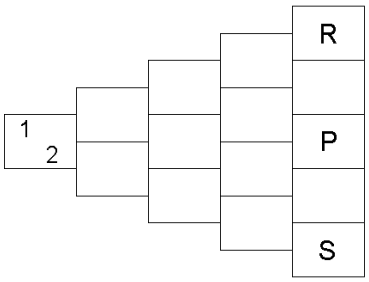


Figure 6.9: Stochastic game version of Shapley's Game (modified from [19]).

	c	d
C	$0, 0$	$0, 1$
D	$1, 0$	$0, 0$

Figure 6.10: Payoff matrix of stage games played in the stochastic version of Shapley's game shown in Figure 6.9.

6.6.3 Shapley's Game

Cook has created a multi-stage version of Shapley's game [19]. A simplified, but essentially equivalent, version of this game is depicted in Figure 6.9. In this game, the two agents (labeled 1 and 2) begin in the left-most column. At each iteration, each agent can take one of two actions: up or down. An up action moves the agent up and to the right, while a down action moves the agent down and to the right. When the agents reach the rightmost column (after four joint actions), payoffs are assigned based on the episodal actions of the agents, which are determined by the label of the cell that each agent occupies (R, P, S or no label) at the end of the episode. Payoffs are assigned as in the game Rock (R) Paper (P) Scissor (S) (except that the *losing* agent receives a payoff of 0 rather than a payoff of -1). If an agent moves to an unlabeled goal state (in the rightmost column), the agent automatically receives a payoff of 0, and the other agent receives a payoff of 1 if it moves to a labeled goal state. Otherwise, it also receives a payoff of 0.

While this game models Shapely's game to some degree, some of its stage games become identical to the matrix game shown in Figure 6.10, which we studied in the previous chapter (see Section 5.6.2). Unlike Shapley's game in which each agent has a minimax value of $\frac{1}{3}$, each agent has a minimax value of 0 in this game. Thus, in this

[10]The second agent only received a payoff of 3 since it learned to "waste" a move in each episode. While this behavior might seem undesirable on the surface, this agent learned that it should do so to achieve the compromise. As a result, it received a higher average net payoff than its minimax value of 2.

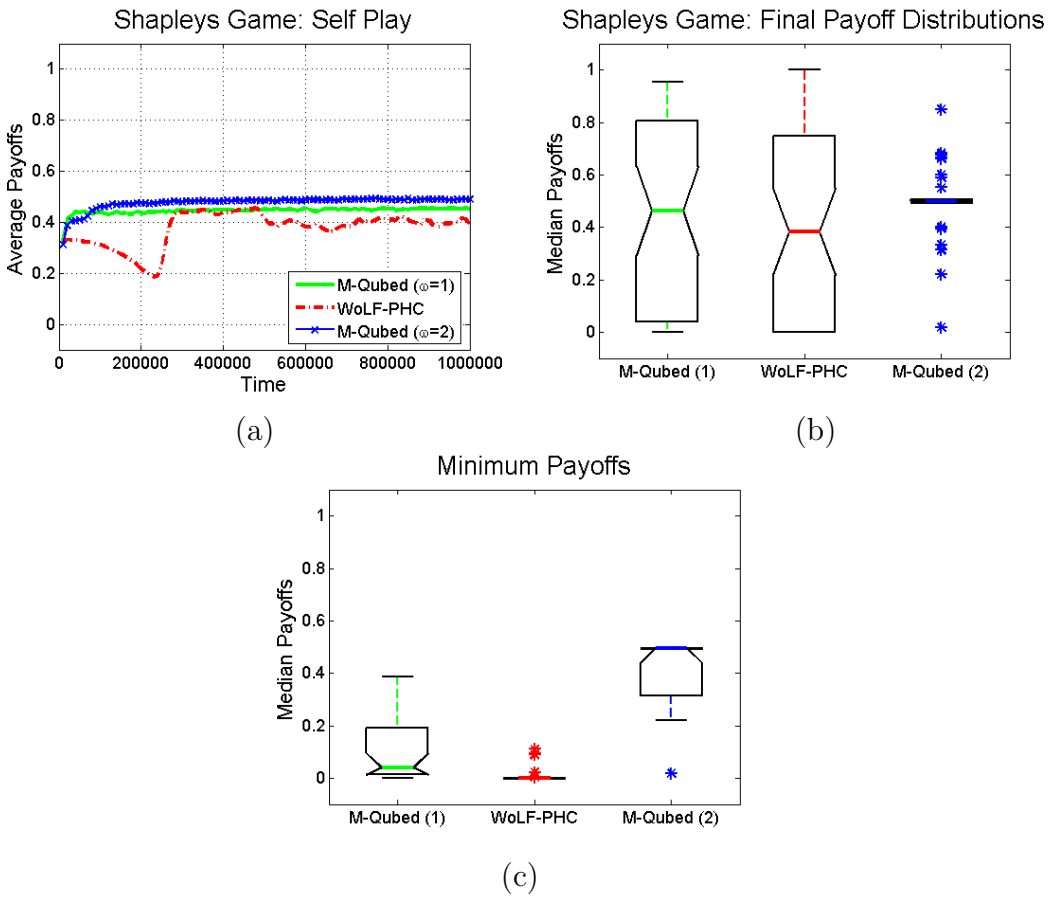


Figure 6.11: (a) Average payoffs over time to the agents (in self play) for Shapley's game depicted in Figure 6.9. Results are an average of 25 trials. (b) Box and whisker plots showing the distribution of the agents' payoffs after 1000000 episodes. (c) The average payoffs to the agent that receives the lowest payoff. In the figures, M-Qubed (1) indicates M-Qubed agents with $\omega_i = 1$ and M-Qubed (2) indicates M-Qubed agents with $\omega_i = 2$.

multi-stage Shapley's game, an agent can guarantee itself a payoff of no more than 0. However, profitable compromises can be reached in this game (as demonstrated in Section 5.6.2).

The performance of M-Qubed and WoLF-PHC in this game is shown in Figure 6.11. M-Qubed with $\omega_i = 1$ (labeled M-Qubed(1) in the figure) receives an average payoff of approximately 0.45 in this game, and WoLF-PHC receives an average payoff of approximately 0.4. However, in both cases, this is a result of one of the agents receiving an average payoff close to 1 on most episodes while the other agent receives an average payoff close to 0 (see Figures 6.11(b) and (c)). Thus, neither of the algorithms learns to play cooperatively consistently, although M-Qubed (1) does reach a compromise on some occasions.

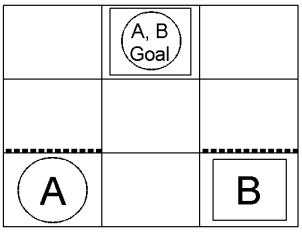


Figure 6.12: Stochastic game version of the game chicken.

However, as we showed in the previous chapter (see Section 5.6.2), if ω_i is increased for each agent i, then M-Qubed usually does learn to compromise so that the agents alternate between winning and losing in this game. Thus, when $\omega_i = 2$ (labeled M-Qubed (2) in the figure), the average payoff to the M-Qubed (2) agent that receives the lower payoff is significantly higher (on average) than for M-Qubed (1) and WoLF-PHC. This result is shown in Figure 6.11(c). In most instances, each of the M-Qubed (2) agents receives an average payoff near 0.5. This results in a higher average payoff to both M-Qubed (2) agents (approximately 0.49).

6.6.4 Chicken

Figure 6.12 depicts a stochastic game related to the matrix game chicken. This game, which has been studied in [9, 52, 39], is similar to the coordination game shown in Figure 6.1 except that a) the agents share the same goal cell (which they are allowed to enter simultaneously) and b) a probabilistic barrier is place above each agent's start position. This barrier stops half of the agents' attempts to move upward from their start states. Thus, it is more profitable (on average) for an agent to move to the center column before moving upward. However, since both agents cannot occupy the same cell simultaneously (other than the goal cell), this action risks a collision with the other agent.

The performance of the agents is shown in Figure 6.13. While M-Qubed learns slightly faster in this game than WoLF-PHC, both algorithms learn to play the same solution. That is, one agent learns to move to the center, while the other learns to test the barrier. This results in a net payoff of 100 to one agent, and 50 to the other (and a combined average of 75). Since 50 is an agent's minimax payoff neither of the agents learns a profitable compromise[11] in this game.

One reason that M-Qubed fails to learn to compromise in this game is because learning restarts make one agent more selective than the other, which results in unbalanced payoffs. One possible way to overcome this is for the agents to coordinate

[11] One such profitable compromise is for the agents to alternate between *winning* (moving to the center) and *losing* (testing the barrier).

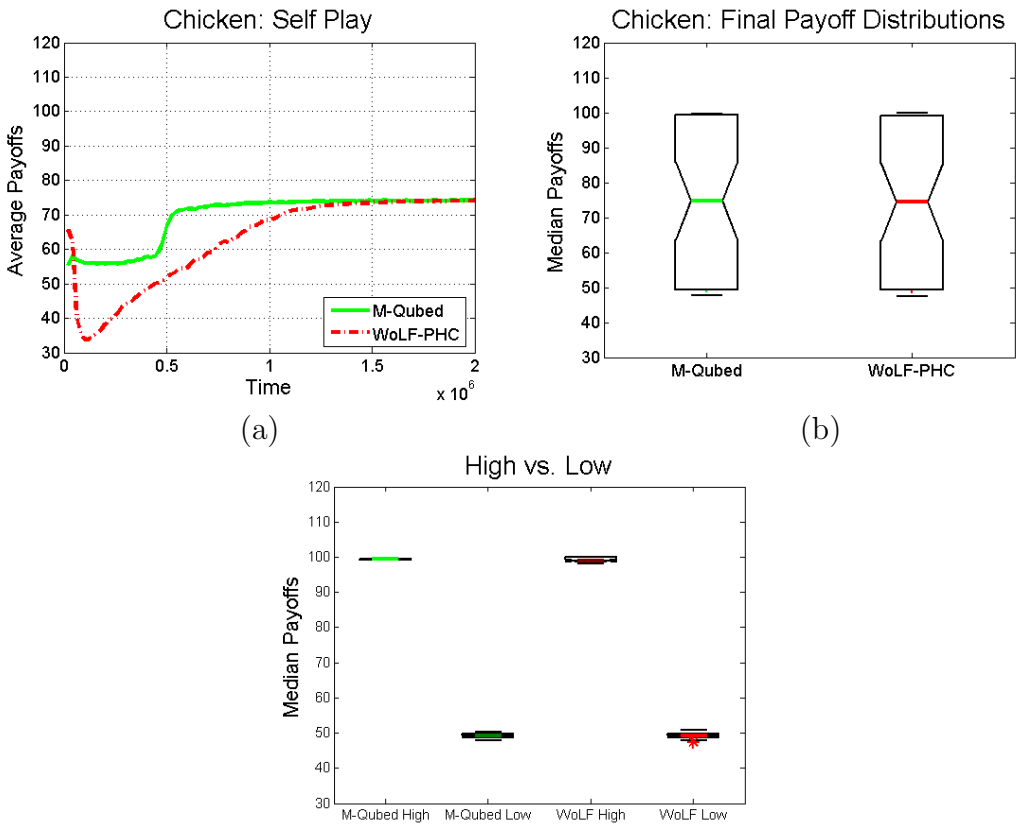


Figure 6.13: (a) Average payoffs over time to the agents (in self play) for the game chicken depicted in Figure 6.12. Results are an average of 25 trials. (b) Box and whisker plots showing the distribution of the agents' payoffs after 2000000 episodes. (c) The average payoffs to the agent receiving the higher and lower payoffs respectively.

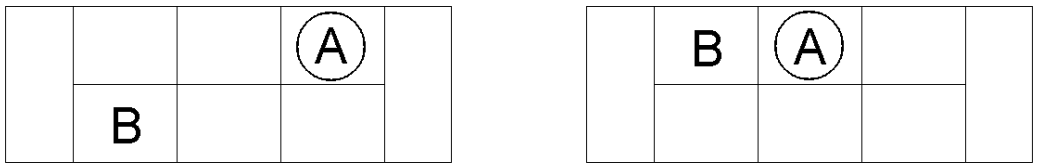


Figure 6.14: A modified version of soccer (the original was designed by Littman in [51]). (Left) the initial board position. (Right) A situation requiring a probabilistic choice for both agents.

trembles (as with SAwT in Chapter 4). However, coordinating trembles is not easily done in the case of M-Qubed without explicit communications. We leave this for future work.

It is possible that M-Qubed is unable to learn to compromise in this game for some of the same reasons that it is unable to compromise in the stochastic version of Shapley's game (see also Section 5.6.2). Thus, increasing ω_i for each agent i could possibly increase M-Qubed's ability to cooperate. However, doing so makes the state space very large, which would result in very slow learning. For this reason, we do not provide this analysis in this disseration. We leave further analysis of the behavior of M-Qubed in this game to future work.

6.6.5 Soccer

All of the stochastic games that we have discussed so far are either cooperative games (in the case of the coordination game) or games of conflicting interest (prisoner's dilemma, chicken, and Shapley's game). We now discuss the performance of M-Qubed in a competitive stochastic game. This game, shown in Figure 6.14, is a modified version of the soccer game presented by Littman in [51]. The initial board position of each episode is depicted in Figure 6.14 (left).

In this game, two players, `A` and `B`, occupy distinct squares and can move in any of the four compass directions or stay in place (stand)[12]. After both players have selected their actions, the two moves are executed in random order.

The circle in the figure represents the "ball." When the player with the ball steps into the appropriate goal (the left-most cell for player `A` and right-most cell for player `B`), that player scores a point (and the episode ends).

When the player with the ball selects an action that would move it into the cell occupied by its opponent, possession of the ball changes. Additionally, if an agent

[12]As we have done for other stochastic games, we eliminated unnecessary actions to increase learning speed.

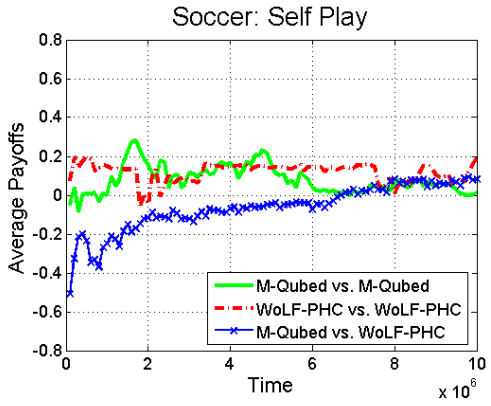


Figure 6.15: Average payoffs to various agents in the game of soccer.

tries to move into the square of a standing agent that has possession of the ball, then possession changes with probability 0.2.

Figure 6.14 (right) shows a situation in this game in which the agents must learn to act probabilistically (when associating with intelligent agents) since a deterministic strategy by either agent in this a situation could easily be exploited.

Figure 6.15 shows the payoffs for a single trial to M-Qubed in self play, WoLF-PHC in self play, and M-Qubed against WoLF-PHC. While M-Qubed learns extremely slowly in this game (due to an overuse of learning resets), it does eventually learn to avoid being exploited in self play and against WoLF-PHC. In fact, M-Qubed eventually learns to exploit WoLF-PHC somewhat in this game. We, note, however, that the figure shows that WoLF-PHC can also exploit itself in self play.

6.7 Summary and Discussion

While the stage game of a stochastic game is a matrix game, it is more difficult to learn than matrix games that we discussed in previous chapters. This is because the payoff matrix of a stage game is non-stationary due to the changing strategies of the learning agents in subsequent stages. Thus, the M-Qubed algorithm presented in the previous chapter is not suitable for learning stage games of stochastic games. In this chapter, we extended the M-Qubed algorithm to learn strategies in the stage games of stochastic games. We demonstrated (empirically) the algorithm's performance in several imporant stochastic games.

In this chapter, we have shown (empirically) that the extended version of M-Qubed demonstrates the security property in self play in several important stochastic games. Additionally, we showed that M-Qubed demonstrates the security property against WoLF-PHC in the stochastic game of soccer. However, while M-Qubed satisfies the security property in all matrix games, it does not appear to do so in all stochastic

games. In games with very large state space and certain probability transition functions, *potential payoffs* might not propogate from stage to stage fast enough. As a result, M-Qubed might still converge (because it stops exploring) prematurely to undesirable policies in certain stages, resulting in low payoffs. We note, however, that if M-Qubed does learn the stage payoffs properly (in the correct order), it does satisfy the security property.

We have also shown that M-Qubed demonstrates the compromise/cooperate property in self play in some important stochastic games. In particular, we showed that M-Qubed satisfies this property in a multi-state prisoner's dilemma and in a multi-state version of Shapley's game. Multiagent learning algorithms in the literature do not learn to offer and accept profitable compromises in many of these games. In this chapter, we demonstrated this result for the case of WoLF-PHC, which demonstrates typical behavior of traditional learning algorithms in these games.

While M-Qubed appears to be the first learning algorithm of its kind to offer and accept profitable compromises in some theoretically important stochastic games, there is much potential for future work. First, while each free parameter of the extended algorithm is used to overcome specific challenges, it would be useful to reduce the number of free parameters. Second, M-Qubed learns very slowly in many stochastic games. This is caused by several factors including learning restarts, dependence on first-visit Monte Carlo methods (which do not effectively use much of the data that the agent gathers via experience), and small parameter updates (when the credibility of a sample is low). Additionally, M-Qubed's state space (like most reinforcement learning algorithms) grows exponentially with increasing numbers of states, actions, and agents.

Chapter 7

Summary and Future Work

7.1 Introduction

In repeated general-sum games, the goal of an agent is to maximize its own payoffs over time. However, most existing multiagent learning algorithms found in the literature learn to play myopically when associating with other learning agents. As a result, agents employing these algorithms often receive low payoffs in many important general-sum games. In this dissertation, we analyzed and developed multiagent learning algorithms that behave less-myopically in many of these games. Thus, these algorithms receive higher average payoffs than those found in the literature.

In this chapter, we summarize these results and present future work.

7.2 Summary

In Chapters 3 and 4, we considered the case in which an agent can observe its own actions and payoffs, but it cannot observe the actions and payoffs of other agents. The S-Algorithm, developed by Karandikar *et al.* [46] and Stimpson *et al.* [71, 70], is an aspiration-based satisficing learning algorithm for such situations. Stimpson *et al.* analyzed this algorithm in a particular multiagent social dilemma, and found that it converges in this game (in self play) to pareto efficient solutions with high probability given that certain conditions are satisfied. In Chapter 3, we analyzed the S-Algorithm in general-sum matrix games, and showed that it converges with high probability to near pareto efficient solutions (in self play) in most of these games (and all 2-player matrix games), provided, again, that certain conditions are met. In Chapter 4, we extended the S-Algorithm to include aspiration trembles. We showed that this extended algorithm can learn to tremble its aspirations over time to increase its own long-term payoffs. Additionally, this algorithm (called SAwT) eliminates the S-Algorithm's dependence on initial parameter settings (for convergence to pareto efficient solutions).

We also considered the case in which an agent can, in addition to observing its own actions and payoffs, observe the actions of other agents. Given such information, an agent should learn to satisfy two properties. First, an agent should never learn to play strategies that produce average payoffs less than the minimax value of the game. Second, in noncompetitive games, an agent should learn to offer and accept

profitable compromises (when possible) so that it receives average payoffs higher than its minimax value. We present an algorithm (called M-Qubed) in Chapter 5 that provably satisfies the first property (regardless of its associate) in all general-sum matrix games, and empirically displays the second property in self play of many important matrix games. We know of no algorithm in the literature that satisfies these two properties simultaneously in general-sum games under these knowledge restrictions.

In Chapter 6, we consider learning in repeated stochastic games. Stochastic games generalize matrix games and are much more difficult for agents to learn. No algorithm presented in the literature effectively learns in many of these games. In this chapter, we extended the M-Qubed algorithm so that it can learn effectively in these games. We showed (empirically) that M-Qubed possesses these same two properties in some important general-sum stochastic games. Multiagent learning algorithms from the literature do not satisfy these two properties in many of these same games.

7.3 Future Work

Although we believe that the contributions of this dissertation are important, they leave many questions unanswered. In addition to the future work we mentioned at the end of each chapter, we emphasize two important issues that we have left for future work.

First, in the main body of this dissertation, we considered only the specific cases in which a) an agent has full knowledge of its own actions and payoffs (and cannot observe the actions and payoffs of other agents) and b) an agent has full knowledge of the actions of other agents as well as its own actions and payoffs. However, practical applications of multiagent learning involve games in which these variables are only partially observable. Future work should analyze and develop multiagent learning algorithms for such situations.

Additionally, none of the algorithm analyzed in the main body of this dissertation consider the payoffs of other agents. However, evidence shows that people do use the payoffs of other agents to determine how to behave [12]. In the appendix, we present a learning algorithm which also uses knowledge of the payoffs of other agents to learn more effectively. While this work can be considered preliminary, we give empirical results showing that this algorithm learns effectively when associating with a large class of learning agents in 2-player matrix games. In contrast, both the S-Algorithm and M-Qubed struggle to find ways to offer and accept compromises when associating with other kinds of learning agents in some games.

Second, in this dissertation, we considered only infinitely repeated games between the same agents. Many practical applications of multiagent learning, however, involve finitely repeated games between the same agents. This makes the multiagent learning

problem much more difficult since the speed of learning becomes much more important. Additionally, compromises that are profitable in infinitely repeated games may not be profitable in finitely repeated games. Thus, future work should include the development of algorithms that learn non-myopic (yet payoff maximizing) solutions in these games.

Another area of future work is that of multiagent learning in repeated games in which the same agents do not play all episodes of the game. Some of these situations have been studied in detail (see, for example, [33]), but it is typically assumed (particularly if agents are unlikely to meet again) that an agent's actions do not affect the subsequent actions of other agents. However, real world situations dictate that agents can often observe (as onlookers) the actions of future associates. Thus, even in these situations, it cannot be assumed that an agent's actions do not influence the future actions of associates.

Appendix A

Learning to Teach and Follow in Repeated Games

A.1 Introduction

In the main body of this dissertation, we focused on learning in situations in which the payoffs of other agents cannot be observed. We now consider the case in which agents can observe the payoffs of other agents.

Past algorithms designed for repeated games generally fall into one of two categories. The first category of algorithms, which we will call *follower* algorithms, adapt their strategies with the goal of finding a best response to the play of their associate(s). This approach allows an agent to accept good compromises offered by associates. However, follower agents are often unable to offer acceptable compromises to others. Consequently, follower agents often earn low average payoffs over time in many important games. The second category of algorithms, which we will call *teacher* algorithms, seek to "teach" their associate(s) to play acceptable (for the teacher) strategies. These agents are often capable of offering acceptable compromises, but are generally unable to accept profitable compromises suggested by others.

In this paper, we present a technique for combining teacher and follower strategies into a single agent/algorithm. We call this algorithm SPaM (*S*ocial and *Pa*yoff *M*aximization). We show that combining follower and teacher attributes as does SPaM results in an agent receiving high average payoffs over time when associating with a large variety of learning agents (including self, best-response learners, and, perhaps, humans) in many important 2-agent matrix games. Prior preliminary research also suggests that SPaM can be implemented effectively in multi-state repeated games [21].

A.2 Background and Related Work

As we mentioned in the introduction, past work in multiagent learning for repeated general-sum games can be separated into two categories: follower algorithms and teacher algorithms. In this section, we review some of this work.

A.2.1 Follower Algorithms

Development of the theory of follower algorithms (or adaptive learners [12]) has occupied the energy of most research in multiagent learning. Follower approaches

include fictitious play [33], Q-learning [76, 66, 54], Q-learning variants (e.g., [51, 52]), policy gradient methods (e.g., [9]), and no-regret algorithms (e.g., [43, 8]).

The goal of follower algorithms is typically to learn a best response to the assumed stationary strategies of other agents. When all agents play a best response to the strategies of the other agents, the results is a Nash equilibrium (NE). However, achieving this goal has been challenging. Despite some successes, follower algorithms have been mostly unsuccessful at learning non-myopic equilibrium in repeated games.

A.2.2 Teacher Algorithms

Follower algorithms frequently have the characteristic of ignoring the payoffs of other agents in the game. This characteristic is actually necessary in many important contexts since the payoffs (and actions) of other agents may be completely unknown. However, in many other contexts, at least some knowledge about the payoffs of other agents is available. Evidence suggests that humans do use information about the payoffs of other agents when making decisions [12]. Teacher algorithms use this information to coax other agents to play differently.

Perhaps the most famous teacher is tit-for-tat (TFT) [2], which was designed for the iterated prisoner's dilemma. TFT plays the action played by its associate in the previous time period. The desired result is that TFT's associate learns that cooperation yields higher average payoffs over time than does defection. Thus, TFT receives higher payoffs than it would otherwise.

The economics literature provides several important teacher algorithms (e.g. [12]). Some of the earlier work involved reputation formation [47, 56]. These works show that, in repeated market games, a firm may benefit by establishing a "tough" reputation in early rounds, even though establishing such a reputation may produce low immediate payoffs.

Additional teacher algorithms (called leader algorithms) were given by Littman and Stone [54] for 2-player matrix games. These teacher algorithms (called *Bully* and *Godfather*) were each shown to induce desirable behavior from certain kinds of associates in many games. Using *Godfather*, Littman and Stone developed an algorithm (which we will call *Godfather++*) for calculating a repeated-play NE for 2-player matrix games [53]. We now discuss this algorithm in more detail.

Let player i be the agent in question, and let player $-i$ be its associate[1]. Let, $A_j(s)$ be the actions available to player j in state s, and let $a_j^t \in A_j(s)$ be player j's action at time t (from state s). The *joint-action* $a^t = (a_i^t, a_{-i}^t)$ is a vector of the actions taken by the agents at time t.

Godfather++ calculates a target solution (which may require that the agents alternate between joint-actions) which maximizes the product of the agents' (positive)

[1]We prefer the term *associate* to *opponent*, since the agents may not be competing.

advantages[2]. Let c_i^t and c_{-i}^t be the target actions (corresponding to the target solution) for players i and $-i$ at time t, respectively. As long as $a_{-i}^t = c_{-i}^t$, player i plays c_i^{t+1} at time $t+1$. However, if $a_{-i}^t \neq c_{-i}^t$, then player i *punishes* player $-i$ for the next n (calculated from the game matrix) rounds of the game by playing an attack strategy. This results in player $-i$ receiving an expected reward of no more than its minimax value each of the n punishment rounds. Thus, an agent's best response to *Godfather++* is to play c_{-i}^t at each time t.

Godfather++ has some very nice theoretical properties, although it does have several issues that should be addressed. Some of these issues include the following:

1 The length of the punishment phase of *Godfather++* (denoted by the variable n) may be unnecessarily large (as Littman and Stone acknowledge). Long punishment phases make it difficult for a learner to determine which of its action is being punished or rewarded (credit assignment problem). Thus, to increase teaching effectivness, the punishment phase should be as short as possible. Additionally, the majority of learning algorithms in the literature typically only condition utilities on very recent action histories, meaning that these agents are unable to "perceive" long punishment phases.

2 *Godfather++* does not specify how the agents should coordinate their actions when the target solution is a sequence of joint-actions. This means that agents may never play cooperatively in some games.

3 The strategy used by *Godfather++* in the punishment phase (i.e., the attack strategy) does not take into account the agent's own payoffs. Thus, vengeance may be overly costly. While this is not extremely problematic (in the limit) if the associate eventually learns to cooperate, there is no guarantee that the associate will learn to cooperate.

4 The target solution is not guaranteed to be unique. Thus, the agents may be unable to coordinate cooperative behavior (particularly in self play). Additionally, a similar (but different) solution may be proposed (and enforced) by an associate, which the algorithm cannot learn to accept.

5 The algorithm requires complete knowledge of the game matrix, as well as perfect knowledge of all agents' actions.

The algorithm we present next addresses points 1-3 (at least in part). We do not address points 4 and 5.

[2]The *advantages of the agents* are the agents' payoffs minus their minimax value.

1. $S = \{a_i : T(s, a_i) \geq 0\} \cup \{\text{argmax}_a\, T(s, a_i)\}$

2. Select action a_i^t

$$a_i^t = \begin{cases} \text{argmax}_{a_i \in S}\, F(s, a_i) & \text{with probability } 1 - \eta \\ \text{argmax}_{a_i}\, F(s, a_i) & \text{with probability } \rho\eta \\ \text{random} & \text{with probability } (1 - \rho)\eta \end{cases}$$

Algorithm A.1: Action selection in SPaM. In the figure, $T(\cdot)$ and $F(\cdot)$ represent teacher and follower utilities (respectively).

A.3 An Algorithm for Teaching and Following

Littman and Stone [54] concluded that neither teacher nor follower strategies alone are sufficient for successful multiagent learning. Rather, agents should employ mixtures of teacher and follower techniques. It is, however, unclear how teacher and follower strategies can be mixed successfully. One methodology is offered by Powers and Shoham [63, 62], in which an agent uses a teacher strategy (either *Godfather* or *Bully*) in early rounds of the game. After a deterministic number of iterations, the algorithm evaluates whether or not the associate is responding to the teacher strategy. If it is not, the algorithm switches to a follower strategy. However, their approach does not take into account both teacher and follower strategies simultaneously.

In this paper, we propose a methodology for combining follower and teacher strategies. Although incomplete, the methodology offers a first step towards employing follower and teacher strategies simultaneously. We call this algorithm SPaM (*S*ocial and *Pa*yoff *M*aximizing).

SPaM learns both a teacher and a follower (payoff maximizing) utility function. The teacher utility function estimates how well actions induce "good" behavior from associates. The follower utility function, as always, estimates the actual material payoffs that an action yields.

Let $T(s, a_i)$ and $F(s, a_i)$ be (respectively) the teacher and follower utilities for playing action a from state s. SPaM uses $T(s, a_i)$ and $F(s, a_i)$ to select its actions using the constrained maximization technique shown in Algorithm A.1. In words, S is the set of actions that have non-negative teacher utility (if no action has non-negative teacher utility, then S contains only the action with the highest teacher utility). With high probability (i.e., probability $1 - \eta$), SPaM selects the action in the set S with the highest follower utility. With a small probability (i.e., probability $\rho\eta$) SPaM selects the action with the highest follower utility (whether or not it is in the set S). Otherwise, SPaM explores randomly.

We now describe how SPaM learns $T(s, a_i)$ and $F(s, a_i)$ in repeated matrix games[3].

A.3.1 Learning a Teacher Utility Function

The goal of a *teacher* algorithm is to make player $-i$'s best response desirable for the teacher. This entails that an associate should be a) rewarded for playing "good" actions and b) "punished" (if necessary) for deviating from those "good" actions. For "good" actions to be player $-i$'s best response, the punishment must exceed the profit of deviating.

SPaM, like *Godfather++*, determines that "good" actions contribute to a sequence of joint-actions that maximize the product of the agents' (positive) advantages (as justified by Nash [59])[4]. Let this sequence of joint-actions, called the *target solution*, be denoted by c and let $c^t = (c_j^t, c_{-j}^t)$ be the joint-action which c specifies at time t. Also, let $|c|$ denote the length of the sequence of c. In games in which $|c| > 1$, we must determine how agents cycle through the sequence of joint-actions. In our current implementation, $c^t = c^{t-1}$ if c^{t-1} is not played at time $t-1$. Otherwise, the current joint-strategy of the target solution updates as in *Godfather++*[5].

SPaM uses the notion of guilt (denoted G_j^t for player j's guilt at time t) to determine whether an agent should be rewarded or punished. When $G_j^t > 0$, then player j has some level of guilt. When $G_j^t \leq 0$, then player j is *guiltless*. As a rule, a guilty agent should be punished while a guiltless agent should be rewarded. We describe how SPaM determines guilt in the discussion that follows.

Let r_j^t be the payoff to agent j at time t. Also, let $r_j(c^t)$ be the *expected* payoff to agent j when the joint-action c^t is played, and let $r_j(c) = \frac{1}{|c|}\sum_{c^t \in c} r_j(c^t)$ be the *average expected* payoff to player j if both agents play the target solution.

Initially, each agent is guiltless (i.e., $G_j^0 = 0$). Thereafter, the guilt of each player j is updated using the tree shown in Figure A.1, which specifies six unique cases (labeled 1-6). Each of the six cases is represented by a unique path through the tree. For example, case 1 is the situation in which $a_j^t = c_j^t$, $G_j^t > 0$ and $a_{-j}^t = c_{-j}^t$. We now explain and justify how guilt is updated in each case.

1 The joint-action c^t is played when player j is guilty. Since player j is not being punished (while guilty) in this case, its guilt would normally be maintained.

[3]In [21], we described how SPaM can be used in two-player repeated multi-state games.

[4]The maximum possible product of the agents' (positive) advantages in competitive games is zero. In 2-player competitive games, a teacher algorithm has nothing to offer. Thus, $T(s, a) = 1$ for all pairs (s, a), meaning that the teacher utility function has no effect on the behavior of the agent (see Algorithm A.1). The rest of this section deals with the case in which the maximum product of the agents' advantages is nonzero.

[5]This implementation is sufficient for all games discussed in this paper, but should be modified for the general case.

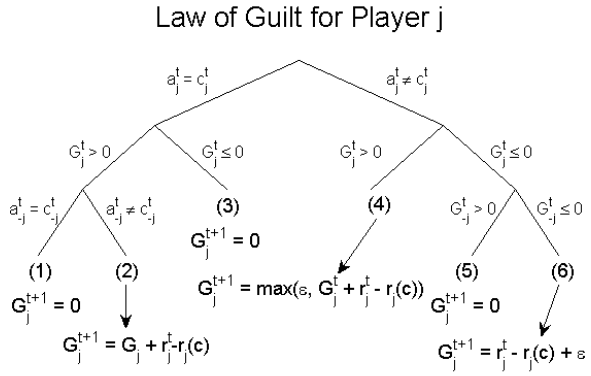


Figure A.1: Tree showing how player j's guilt value (G_j) is updated after each round of the game.

However, in this special case, to avoid the possibility that player j may attribute future punishment to the playing of the target solution, player j's guilt is absolved[6].

2 When player j is guilty and plays cooperatively, its guilt is updated by $G_j^{t+1} = G_j^t + r_j^t - r_j(c)$. This means that player j's guilt is reduced if its payoff (r_j^t) is less than $r_j(c)$. Note that if $r_j(c)$ is at least G_j^t more than r_j^t, then player j's guilt is absolved.

3 Player j follows the target strategy while guiltless. Thus, it remains guiltless.

4 Player j deviates while guilty. Its guilt is updated as in case 2, except that its guilt cannot be completely absolved (since it deviated). Thus, its new guilt is the max of some small value $\varepsilon > 0$ and $G_j^t + r_j^t - r_j(c)$.

5 Player j deviates when its associate is guilty. In this case, player j is justified in deviating (so it remains guiltless).

6 Player j deviates when its associate is guiltless. In this case, player j acquires guilt to the degree that it profits from the deviation (the difference between r_j^t and $r_j(c)$, plus some small value $\varepsilon > 0$[7]). Thus, if deviating yields a profit less than or equal to -ε, an agent remains guiltless.

Note that SPaM measures its own guilt as well as its associate's guilt. Thus, SPaM's associate can hold SPaM accountable in the same way that SPaM holds its

[6]We note that this rule could possibly (in a small set of games) allow deviating to be a best response. We leave this to future work.

[7]ε is added to player j's guilt to require that player j be punished by *more than* it gains from deviating.

associate accountable. In this way, SPaM can (potentially) teach and follow simultaneously.

In short, player j's guilt is the (positive) profit it has obtained from deviating in the past. When guilt is positive, the agent has benefited from deviating and should be punished. Otherwise, the agent should be rewarded (with cooperative actions from its associate). We now specify how a teacher utility function is constructed from the guilt values to obtain the desired behavior.

The action selection methodology in Algorithm A.1 requires that teacher actions have positive utility only if they induce "good" behavior from the associate. Thus, if action a does not serve to induce player $-i$ to play c_{-i} in the future, then $T_t(s, a_i) < 0$. If action a does induce player $-i$ to play c_{-i} in the future, then $T_t(s, a_i) > 0$.

In the case in which player i's associate is guiltless, player i should conform to the target solution. Thus, in this case, the teacher utility $T_t(s, a_i)$ is given by

$$T_t(s, a_i) = \begin{cases} 1 & \text{if } (a_i = c_i^t) \\ -1 & \text{otherwise} \end{cases} \tag{A.1}$$

In the context of Algorithm A.1, Eq. (A.1) assures that, with at least probability $1 - \eta$, SPaM will play c_i^t when $G_{-i}^t \leq 0$.

We now turn to the case in which $G_{-i}^t > 0$. As we have mentioned, in this case, player i should punish player $-i$. Thus, the teacher utility $T_t(s, a_i)$ is a function of how well action a punishes player $-i$. Thus,

$$T_t(s, a_i) = r_{-i}(c^t) - E[U_{-i}(s, a_i | G_{-i}^t > 0)] - E_p \tag{A.2}$$

where $E[U_{-i}(s, a_i | G_{-i}^t > 0)]$ is the expected payoff to player $-i$ based on the empirical distribution of past actions taken by both agents (when player $-i$ is guilty) and E_p is the expected level of punishment.

We make the following observations about Eq. A.2. First, $E[U_{-i}(s, a_i | G_{-i}^t > 0)]$ is the value of the teacher utility function that is learned. Second, when the difference $r_{-i}(c^t) - E[U_{-i}(s, a_i | G_{-i}^t > 0)]$ is greater than E_p, then $T_t(s, a_i) > 0$. Third, when multiple actions produce satisfactory punishments, SPaM will select the least costly of those actions (see Algorithm A.1). Therefore, SPaM can lower its own loses while punishing its associates (see point 3 of the issues discussed in Section A.2).

We discussed in Section A.2 that the punishment phase should be brief (if possible). The variable E_p (the expected punishment level) dictates the duration of the punishment phase. For quick punishments, we use

$$E_p = \min(G_{-i}^t, r_{-i}(c^t) - m_{-i}), \tag{A.3}$$

where m_{-i} is player $-i$'s minimax value. We use this value of E_p since a) player $-i$ need not be punished by more than its guilt value G_{-i}^t and b) since player $-i$ can guarantee itself a payoff of m_{-i} by playing its minimax strategy, an agent should not expect to punish its associate by more than $r_{-i}(c^t) - m_{-i}$.

1) Observe the game matrix; compute c
2) $G_i = G_{-i} = 0$, $T(s, a_i) = 0$, $F(s, a_i) = 0$

3) Repeat,
  a) Take action a_i according to Algorithm A.1
  b) Observe reward r and actions a_i^t, a_{-i}^t:
    i) Update G_i and G_{-i} using Figure A.1
    ii) Updated $U_{-i}(s, a_i | G_{-i} > 0)$
    iii) if $G_{-i} \leq 0$
      compute $T(s, a_i)$ using Eq A.1
      else compute $T(s, a_i)$ using Eq A.2
    iv) Compute $F(s, a_i)$

Algorithm A.2: The complete SPaM algorithm.

A.3.2 Follower Utility Function

The follower algorithm used by SPaM can be any follower algorithm. However, the weaknesses of the algorithm chosen reflect on the performance of SPaM. In general, we favor a learning algorithm which learns quickly, thus, we use a variant of fictitious play [33] (which we call FP). FP varies from standard fictitious play in that it looks ahead an extra time period (rather than just the current time) in determining the action that maximize expected payoffs. Additionally, the previous joint-action taken by the agents is used for state s (as in the teacher utility function).

A.3.3 Algorithm Review

The complete algorithm is sketched in Algorithm A.2. As we mentioned in Section A.2, the algorithm empirically exhibits the following desirable characteristics. First, the length of the punishment phase is minimized provided that $E[U_{-i}(s, a_i | G_{-i}^t > 0)]$ is accurate. As an example, whereas *Godfather++* requires 3 rounds of punishment in the game *battle of the sexes*, SPaM typically requires no rounds of punishment, although one or two rounds of punishment are sometimes necessary. Second, SPaM is able to successfully coordinate profitable cooperative actions even in non-deterministic environments. Third, when punishing an associate, SPaM may select between multiple punishing actions (provided that multiple punishing actions have been shown to be effective). This allows SPaM to select the action that is most profitable to it. SPaM empirically exhibits these properties while maintaining the other positive characteristics of *Godfather++* (it appears).

	a	b
a	3, 3	0, 5
b	5, 0	1, 1

(a)

	a	b
a	4, 4	2, 5
b	5, 2	0, 0

(b)

	a	b
a	0, 3	3, 2
b	1, 0	2, 1

(c)

Figure A.2: Payoff matrices for (a) prisoner's dilemma, (b) chicken, and (c) tricky game [8]. Payoffs to the row player are given first, followed by the payoffs to the column player.

A.4 Results

SPaM performs well when associating with a large variety of agents. We now present some results.

A.4.1 With Automated Agents

Figure A.3 shows the mean average payoffs when SPaM, FP, and WoLF-PHC [9] are matched with each other in the three repeated matrix games show in Figure A.2. In each game, the intended actions of all the agents were executed with probability 0.95 and the opposite action was executed with probability 0.05.

Prisoner's Dilemma. Figures A.3(a)–(c) show results from the iterated prisoner's dilemma (see Figure A.2a). In this game, the target solution is for each agent to play a. Although action a is dominated by action b (for both agents), SPaM is able to teach its associates to play a, resulting in an average payoff per iteration of nearly 3. In this game, SPaM significantly outperforms FP and WoLF-PHC when matched with each of the three agents.

Chicken. Figures A.3(d)–(f) show results from the game chicken (see Figure A.2b). In chicken, the target solution is for each agent to play a, despite the fact that each agent has an incentive to unilaterally deviate. Again, SPaM teaches each of the three associates to do so, resulting in an average payoff per iteration of nearly 4. Again, SPaM outperforms FP and WoLF-PHC in this game when playing with each of the three associates.

Tricky Game. Figures A.3(g)–(l) show results when the agents are matched with each other in tricky game (see Figure A.2c). In tricky game, the target solution is for the row player to play a and the column player to play b, which results in payoffs of 3 and 2 to the row and column players respectively. While the prisoner's dilemma and chicken both generally require only a single round of punishment after a deviation, tricky game frequently requires two rounds of punishment after a deviation. Since our implementations of both FP and WoLF-PHC encode state s as the previous joint-action of the agents, none of the agents are able to "perceive" some of SPaM's

punishments. However, SPaM still performs better than both WoLF-PHC and FP when matched with all three associates, both as a row player and a column player. However, its payoffs are not as high as they otherwise would be had each of the agents used longer histories for state.

A.4.2 With Humans

Preliminary results suggests that SPaM performs very well when playing some games with humans [23, 21]. Future work should more thoroughly explore the performance of SPaM when associating with humans in general-sum games.

A.5 Discussion and Future Work

We have described some of our work in progress on combining teaching and following strategies using SPaM. While as yet incomplete, results show that SPaM performs well in many 2-player matrix games when matched with a wide variety of agents, include itself, follower algorithms, and (in preliminary results not shown here) humans.

Although promising, SPaM suffers from a number of weaknesses. One of these weaknessses is that it does not perform as well as desired when matched with static agents, since static agents cannot be "taught." One possible method for overcoming this challenge is to increase the parameter η when the associate does not respond to teaching. However, we have not as yet found a way to do this effectively without decreasing average payoffs when matched with agents that learn slowly. Additionally, there remains much progress to be made before these these techniques can be used in practical environments (i.e., multi-state environments, environments with incomplete information, and environments with more than two agents).

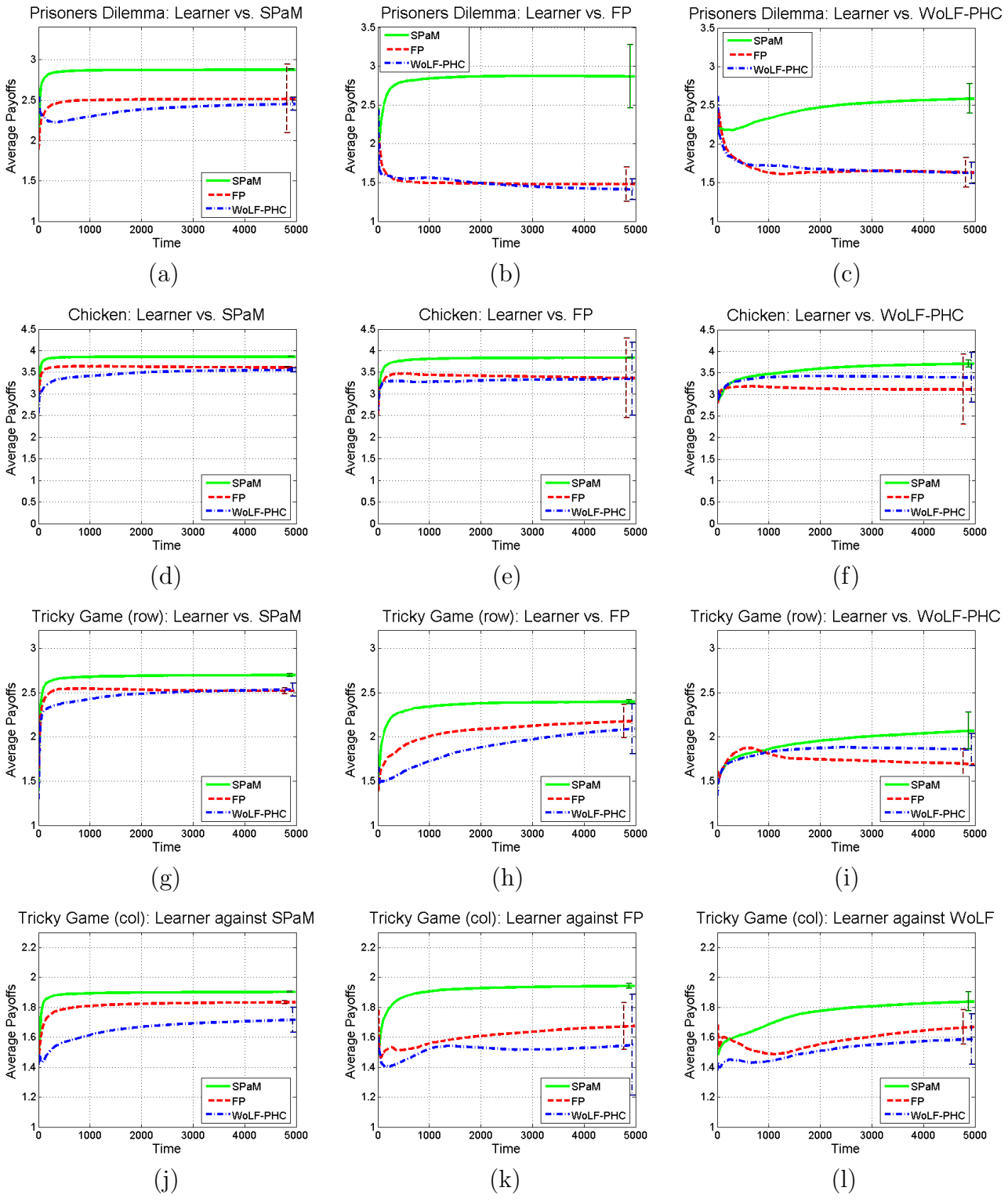


Figure A.3: Payoffs given different pairings of artificial agents. The figures plot the mean (of 50 trials) average payoff per iteration against time. One standard deviation is shown after 5000 iterations. (g)–(i) show the payoffs to the row player in tricky game and (j)–(l) show payoffs to the column player. For SPaM, $\eta = 0.1$ and $\rho = 0.8$.

Bibliography

[1] Robert J. Aumann. Correlated equilibrium as an expression of Bayesian rationality. *Econometrica*, 55(1):1–18, 1987.

[2] Robert Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[3] Forest Baker and Howard Rachlin. Teaching and learning in a probabilistic prisoner's dilemma. *Behavioral Processes*, 57:211–226, 2002.

[4] Bikramjit Banerjee and Jing Peng. Adaptive policy gradient in multiagent learning. In *Proceedings of the 2^{nd} International Joint Conference on Autonomous agents and multiagent systems*, pages 686–692, 2003.

[5] Bikramjit Banerjee and Jing Peng. The role of reactivity in multiagent learning. In *Proceedings of the 3^{rd} International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 538–545, 2004.

[6] Andrea Bonarini, Alessandro Lazaric, Enrique Munoz de Cote, and Marcello Restelli. Improving cooperation among self-interested reinforcement learning agents. In *ECML workshop on Reinforcement learning in non-stationary environments*, Porto, Portugal, October 2005.

[7] Michael Bowling. Convergence problems of general-sum multiagent reinforcement learning. In *Proceedings of the 17^{th} International Conference on Machine Learning*, pages 89–94, 2000.

[8] Michael Bowling. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems 17*, pages 209–216, 2005.

[9] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[10] Steven J. Brahms. *Theory of Moves*. Cambridge University Press, 1994.

[11] Colin F. Camerer. *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton University Press, 2003.

[12] Colin F. Camerer, Teck-Hua Ho, and Juin-Kuan Chong. Sophisticated ewa learning and strategic teaching in repeated games. *Journal of Economic Theory*, 104:137–188, 2002.

[13] John L. Casti. *Five Golden Rules: Great Theories of 20th-Century Mathematics and Why They Matter*. John Wiley & Sons, Inc., 1996.

[14] Yu-Han Chang and Leslie Pack Kaelbling. Playing is believing: The role of beliefs in multi-agent learning. In *Advances in Neural Information Processing Systems 14*, pages 1483–1490, 2002.

[15] Yu-Han Chang and Leslie Pack Kaelbling. Hedge learning: regret-minimization with learning experts. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 121–128, 2005.

[16] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 746–752, 1998.

[17] Andrew M. Colman. Cooperation, psychological game theory, and limitations of rationality in social interaction. *Behavioral and Brain Sciences*, 26:139–198, 2003.

[18] Vincent Conitzer and Tuomas Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *Proceedings of the 20th International Conference on Machine Learning*, pages 83–90, 2003.

[19] Philip Cook. Limitations and extensions of the wolf-phc algorithm. Master's thesis, Brigham Young University, 2006.

[20] Jacob W. Crandall and Michael A. Goodrich. Learning to compete, compromise, and cooperate in repeated general-sum games. In *Proceedings of the 22*$^{\text{nd}}$ *International Conference on Machine Learning*, pages 161–168, Bonn, Germany, 2005.

[21] Jacob W. Crandall and Michael A. Goodrich. Establishing reputation using social commitment in repeated games. In *AAMAS workshop on Learning and Evolution in Agent Based Systems*, New York City, NY, July 2004.

[22] Jacob W. Crandall and Michael A. Goodrich. Learning to teach and follow in repeated games. In *AAAI workshop on Multiagent Learning*, Pittsburgh, PA, July 2005.

[23] Jacob W. Crandall and Michael A. Goodrich. Multiagent learning during ongoing human-machine interactions: The role of reputation. In *AAAI Spring Symposium on Interaction between Humans and Autonomous Systems over Extended Operation*, Stanford, CA, March 2004.

[24] Jacob W. Crandall and Michael A. Goodrich. Learning near-pareto efficient solutions with minimal knowledge requirements using satisficing. In *AAAI Spring Symp. on Artificial Multiagent Learning*, Washington, DC, October 2004.

[25] Antonio Damasio. *Descartes' Error: Emotion, Reason, and the Human Brain*. G.P. Putnam's Sons, 1994.

[26] Antonio Damasio. *The Feeling of What Happens: Body and Emotion in the Making of Consciousness*. Harcourt Brace & Company, 1999.

[27] Daniela de Farias and Nimrod Megiddo. How to combine expert (or novice) advice when actions impact the environment. In *Advances in Neural Information Processing Systems 16*, 2004.

[28] Daniela de Farias and Nimrod Megiddo. Exploration-exploitation tradeoffs for expert algorithms in reactive environments. In *Advances in Neural Information Processing Systems 17*, pages 409–416, 2005.

[29] Richard Dearden, Nir Friedman, and Stuart J. Russell. Bayesian Q-learning. In *Proceedings of the 15^{th} National Conference on Artificial Intelligence*, pages 761–768, 1998.

[30] Jane J. Mansbride (Editor). *Beyond Self Interest*. The University of Chicago Press, 1990.

[31] Dean P. Foster and Rakesh Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29:7–35, 1999.

[32] Robert H. Frank. *Passions Within Reason: The Strategic Role of the Emotions*. W. W. Norton and Company, 1988.

[33] Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. The MIT Press, 1998.

[34] Gerd Gigerenzer, Peter M. Todd, and the ABC research group. *Simple Heuristics that make us smart*. Oxford University Press, 1999.

[35] Herbert Gintis. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Behavior*. Princeton University Press, 2000.

[36] Michael A. Goodrich, Jacob W. Crandall, and Jeff R. Stimpson. Neglect tolerant teaming: Issues and dilemmas. In *AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments*, Stanford, CA, March 2003.

[37] Michael A. Goodrich and Morgan Quigley. Satisficing Q-learning: Efficient learning in problems with dichotomous attributes. In *Proceedings of the International Conference on Machine Learning and Applications*, 2004.

[38] Michael A. Goodrich, Wynn C. Stirling, and Robert L. Frost. A theory of satisficing decisions and control. *IEEE Transactions on Systems, Man, and Cybernetics — Part A: Systems and Humans*, 28(6):763–779, 1998.

[39] Amy Greenwald and Keith Hall. Correlated Q-learning. In *Proceedings of the 20^{th} International Conference on Machine Learning*, pages 242–249, 2003.

[40] Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[41] Junling Hu and Michael P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the 15^{th} International Conference on Machine Learning*, pages 242–250, 1998.

[42] Junling Hu and Michael P. Wellman. Experimental results on Q-learning for general-sum stochastic games. In *Proceedings of the 17^{th} International Conference on Machine Learning*, pages 407–414, 2000.

[43] Amir Jafari, Amy Greenwald, David Gondek, and Gunes Ercal. On no-regret learning, fictitious play, and nash equilibrium. In *Proceedings of the 18^{th} International Conference on Machine Learning*, pages 226–233, 2001.

[44] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–277, 1996.

[45] Ehud Kalai and Ehud Lehrer. Rational learning leads to nash equilibrium. *Econometrica*, 61(5):1019–1045, 1993.

[46] Rajeeva Karandikar, Dilip Mookherjee, Debraj Ray, and Fernando Vega-Redondo. Evolving aspirations and cooperation. *Journal of Economic Theory*, 80:292–331, 1998.

[47] David M. Kreps and Robert Wilson. Reputation and imperfect information. *Journal of Economic Theory*, 27:253–279, 1982.

[48] Isaac Levi. *Hard Choices*. Cambridge University Press, 1986.

[49] Isaac Levi. *The Covenant of Reason*. Cambridge University Press, 1987.

[50] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:(2):212–261, 1992.

[51] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 157–163, 1994.

[52] Michael L. Littman. Friend-or-foe: Q-learning in general-sum games. In *Proceedings of the 18th International Conference on Machine Learning*, pages 322–328, 2001.

[53] Michael L. Littman and Peter Stone. A polynomial-time nash equilibrium algorithm for repeated games. *Decision Support Systems*, 39:55–66, 2005.

[54] Michael L. Littman and Peter Stone. Leading best-response strategies in repeated games. In *IJCAI workshop on Economic Agents, Models, and Mechanisms*, Seattle, WA, August 2001.

[55] Michael L. Littman and Csaba Szepesvari. A generalized reinforcement-learning model: Convergence and applications. In *Proceedings of the 13th International Conference on Machine Learning*, pages 310–318, 1996.

[56] Paul Milgrom and John Roberts. Predation, reputation and entry deterrence. *Journal of Economic Theory*, 27:280–312, 1982.

[57] John Moody, Yefeng Liu, Matthew Saffell, and Kyoungju Youn. Stochastic direct reinforcement. In *AAAI Spring Symposium on Artificial Multiagent Learning*, Washington, DC, October 2004.

[58] Todd K. Moon and Wynn C. Stirling. Satisficing negotiation for resource allocation with disputed resources. In *Negotiation Methods for Autonomous Cooperative Systems: Proceedings from the 2001 AAAI Fall Symposium, Technical Report FS-01-03*, 2003.

[59] John F. Nash. The bargaining problem. *Econometrica*, 28:155–162, 1950.

[60] John F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[61] Christos H. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 749–753, 2001.

[62] Rob Powers and Yoav Shoham. Learning against opponents with bounded memory. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 817–822, 2005.

[63] Rob Powers and Yoav Shoham. New criteria and a new algorithm for learning in multi-agent systems. In *Advances in Neural Information Processing Systems 17*, pages 1089–1096, 2005.

[64] Howard Rachlin. Rational thought and rational behavior: A review of bounded rationality: The adaptive toolbox. *Journal of the Experimental Analysis of Behavior*, 79:409–412, 2003.

[65] Federico Sanabria, Forest Baker, and Howard Rachlin. Learning by pigeons playing against tit-for-tat in an operant prisoner's dilemma. *Learning & Behavior*, 31(4):318–331, 2003.

[66] Tuomas W. Sandholm and Robert H. Crites. Multiagent Reinforcement Learning in the Iterated Prisoner's Dilemma. *Biosystems*, 37:147–166, 1996.

[67] Sandip Sen, Stephane Airiau, and Rajatish Mukherjee. Towards a pareto-optimal solution in general-sum games. In *Proceedings of the 2^{nd} International Conference on Autonomous Agents and Multi-agent Systems*, pages 153–160, 2003.

[68] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, 1996, Third Edition.

[69] Brian Skyrms. *The Stag Hunt and the Evolution of Social Structure*. Cambridge University Press, 2004.

[70] Jeffrey R. Stimpson. Satisficing and cooperation in multiagent social dilemmas. Master's thesis, Brigham Young University, 2002.

[71] Jeffrey R. Stimpson and Michael A. Goodrich. Learning to cooperate in a social dilemma: A satisficing approach to bargaining. In *Proceedings of the 20^{th} International Conference on Machine Learning*, pages 728–735, 2003.

[72] Jeffrey R. Stimpson, Michael A. Goodrich, and Lawrence C. Walters. Satisficing and learning cooperation in the prisoner's dilemma. In *Proceedings of the 17^{th} International Joint Conference on Artificial Intelligence*, pages 535–544, 2001.

[73] Wynn C. Stirling. Games machines play. *Minds and Machines*, 12:327–352, 2002.

[74] Wynn C. Stirling. *Satisficing Games and Decision Making*. Cambridge University Press, 2003.

[75] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[76] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.