Multiagent Learning in the Presence of Agents with Limitations

Michael Bowling

May 14, 2003

CMU-CS-03-118

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
Manuela Veloso, Chair
Avrim Blum
Tuomas Sandholm
Craig Boutilier (University of Toronto)

This research was supported by United States Air Force Grants Nos. F30602-98-2-0135 and F30602-00-2-0549 and by United States Army Grant No. DABT63-99-1-0013. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, by the United States Air Force, the United States Army, or the US Government.

Keywords: machine learning, multiagent systems, game theory, reinforcement learning, multiagent learning, policy gradient methods, non-stationary environments, variable learning rate, game-playing, robot soccer

Abstract

Learning to act in a multiagent environment is a challenging problem. Optimal behavior for one agent depends upon the behavior of the other agents, which are learning as well. Multiagent environments are therefore non-stationary, violating the traditional assumption underlying single-agent learning. In addition, agents in complex tasks may have limitations, such as physical constraints or designer-imposed approximations of the task that make learning tractable. Limitations prevent agents from acting optimally, which complicates the already challenging problem. A learning agent must effectively compensate for its own limitations while exploiting the limitations of the other agents. My thesis research focuses on these two challenges, namely multiagent learning and limitations, and includes four main contributions.

First, the thesis introduces the novel concepts of a variable learning rate and the WoLF (Win or Learn Fast) principle to account for other learning agents. The WoLF principle is capable of making rational learning algorithms converge to optimal policies, and by doing so achieves two properties, rationality and convergence, which had not been achieved by previous techniques. The converging effect of WoLF is proven for a class of matrix games, and demonstrated empirically for a wide-range of stochastic games.

Second, the thesis contributes an analysis of the effect of limitations on the game-theoretic concept of Nash equilibria. The existence of equilibria is important if multiagent learning techniques, which often depend on the concept, are to be applied to realistic problems where limitations are unavoidable. The thesis introduces a general model for the effect of limitations on agent behavior, which is used to analyze the resulting impact on equilibria. The thesis shows that equilibria do exist for a few restricted classes of games and limitations, but even well-behaved limitations do not preserve the existence of equilibria, in general.

Third, the thesis introduces GraWoLF, a general-purpose, scalable, multiagent learning algorithm. GraWoLF combines policy gradient learning techniques with the WoLF variable learning rate. The effectiveness of the learning algorithm is demonstrated in both a card game with an intractably large state space, and an adversarial robot task. These two tasks are complex and agent limitations are prevalent in both.

Fourth, the thesis describes the CMDragons robot soccer team strategy for adapting to an unknown opponent. The strategy uses a notion of plays as coordinated team plans. The selection of team plans is the decision point for adapting the team to its current opponent, based on the outcome of previously executed plays. The CMDragons were the first RoboCup robot team to employ online learning to autonomously alter its behavior during the course of a game.

These four contributions demonstrate that it is possible to effectively learn to act in the presence of other learning agents in complex domains when agents may have limitations. The introduced learning techniques are proven effective in a class of small games, and demonstrated empirically across a wide range of settings that increase in complexity.

Acknowledgments

There are many people who I am deeply indebted to, and without whom this work would never have been finished. First and foremost is my advisor, Manuela Veloso. She has served all of the roles of teacher, mentor, guide, encourager, task-master, advocate, and friend. She has been instrumental in directing me from my initial explorations on this topic to the final thesis revisions. I can only hope that I will be able to advise my future students with a fraction of her skill and patience.

I thank my other committee members, Avrim Blum, Craig Boutilier, and Tuomas Sandholm, for their valuable questions, insights, and guidance. They have served to greatly improve the content and presentation of this work. I also owe gratitude to the late Herb Simon, originally a member of my committee, for his gracious time and pointed questions as this research began. I can only hope that he would have approved of the dissertation that has resulted from those discussions.

There are many people who have played other significant roles along the way. Will Uther, Martin Zinkevich, and Pat Riley participated in numerous conversations and contributed to refining many of the ideas presented here. David Leslie at the University of Bristol put a great deal of effort into replicating some of the results in this thesis. His many questions and patience has improved the detail of the presentation. I have also had the privilege of working closely with Rune Jensen, Gal Kaminka, and Peter Stone on related research topics. Their ideas have undoubtedly influenced those that are contained in this thesis.

The line of research taken in this dissertation was primarily inspired by my experiences on robot soccer teams at CMU. There is a whole army of scientists, graduate students, post-doctoral fellows, and undergraduates, under the direction of Manuela Veloso, that have made this experience possible. In particular, Brett Browning and Jim Bruce were instrumental in developing and building the CMDragons robot soccer team. Without their expertise, one of the key platforms for my thesis research would not have existed. Undergraduates, Allen Chang and Dinesh Govindaraju, also contributed to the team and helped to formally evaluate parts of the system discussed in this work. I would like to thank the many other members of the Multirobot Lab and all of the CMU robot soccer teams, both past and present. In particular, Kwun Han, Sorin Achim, and Peter Stone were all co-members of my first team, the world champion CMUnited '98 small-size team. In addition to those already mentioned, Scott Lenser, Doug Vail, and Sonia Chernova have been fixtures in the lab and at competitions. With these and many others, I have enjoyed the privilege of sharing both research ideas and sleepless nights at competitions. I am thankful to have been a part of CMU's continuing success in robot soccer and acknowledge the influence that all of the members have had on my work.

Equally important were the many family and friends whose encouragement and timely distractions kept me sane, productive, and allowed me to thoroughly enjoy my time as a graduate student. In particular, I would like to thank those with whom I have shared an office over the years:

Will Marrero, Carsten Schuermann, Peter Venable, Dominic Mazzoni, Hyung-Ah Kim, and Kedar Dhamdhere. I have also been blessed with many other caring friends who have offered no end of encouragement. I am deeply indebted to Tom, Theresa, Toby, Chloe, Oliver, Maggie, Jen, and Randy, who have gone above and beyond the call of friend. I have enjoyed starting each week with them filling my house for board games, movies, or the infamous reading of *The Hobbit*. Sunday nights with them made the beginning of the week, and the research that was to be done, much more enjoyable. I would also like to thank my parents, Donald and Joan, and my brother, Jeffery. They have been supportive at every step, always encouraging me to pursue what I am capable of.

Most of all, I would like to thank my wonderful wife, Shayna. Without her support, encouragement, and confidence in me from the day I applied to graduate school until now, this thesis would certainly have gone unwritten. She is my helper and friend, and deserves more credit than I can put into words. It says in Proverbs 18:22, “He who finds a wife, finds a *good thing*.” Shayna is my *good thing*.

Ours is but a small matter in the great deeds of this time. A vain pursuit from its beginning, maybe, which no choice of mine can mar or mend. Well, I have chosen. So let us use the time as best we may!
Aragorn
***The Two Towers* by J. R. R. Tolkien**

Do not regret your choice in the valley of Emyn Muil, nor call it a vain pursuit. You chose amid doubts the path that seemed right: the choice was just, and it has been rewarded.
Gandalf to Aragorn
***The Two Towers* by J. R. R. Tolkien**

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Applications involving or requiring multiple agents (e.g., robot soccer, search and rescue, automated driving, auctions and electronic commerce, information agents) are becoming commonplace as physical robots and software agents become more pervasive. This dissertation focuses on the problem of an agent[1] learning a successful course of action in domains that involve other external learning agents.

An *agent* is the combination of three components: perception, reasoning and action. An agent receives observations about the state of the environment, and chooses an action from those available to the agent. The reasoning component is responsible for mapping the received percepts into a choice of action. Agents commonly have some goal. This may be a desirable state of the environment to achieve, or a signal to maximize. This work focuses on *learning* as part of the agent's reasoning component. Learning in agents involves adapting the mapping of observations to actions through interaction with the environment. Since agents are goal directed, learning seeks to adapt this mapping from percepts to actions in order to improve the agent's goal achievement.

Learning has been studied extensively in single-agent tasks where a robot is acting alone in an unchanging environment. There are a number of advantages of employing learning agents in these tasks. Learning greatly simplifies the problem of agent programming by removing the developmental burden of accurate models and optimal reasoning with respect to those models. Learning also allows a robot to adapt to unforeseen difficulties or changes in the environment. In multiagent environments, learning in agents is both more important and more difficult.

In multiagent domains, agents are forced to interact with other agents, which may have independent goals, assumptions, algorithms, and conventions. In order for agents to handle these environments, they must employ some ability to adapt to the other agents' behavior. Since the other agents are also adapting, this presents a very difficult learning problem that violates the basic stationarity assumption of traditional techniques for behavior learning. Even defining a desirable policy for the agent is difficult, since it depends on the policies of the other agents, which are also improving through experience. This thesis contributes and evaluates learning algorithms that learn in complex domains in the presence of other learning agents.

Realistically, agents have *limitations* and so are not always capable of acting optimally. They

[1]Since this work focuses on the multiagent environments using the framework of stochastic games, we will use the terms robot, agent, player, and even opponent interchangeably. As this dissertation demonstrates, these techniques are applicable to both software and physical agents, and for a variety of competitive and non-competitive domains.

may have physical limitations (e.g., broken actuators or partial observability) that make them incapable of certain behavior. They also may be employing approximations or abstractions of the task in their learning, and so sacrifice optimality in order to learn more quickly. They may not even be learning at all. Limitations are unavoidable in large and complex environments, and create scenarios where the other agents involved may not appear to be acting rationally. Multiagent learning techniques that are practical must address the possibility of limitations, both their own and others'. Effective learning agents should be capable of compensating for their own limitations and those of their teammates, while exploiting limitations in other, possibly adversarial, agents. This thesis also examines how limitations impact learning, and demonstrates effective learning techniques for accounting for limitations in agent behavior.

1.1 Objective

This thesis seeks to answer the question,

> Can an agent effectively *learn* to act in the presence of *other learning agents* in *complex* domains when agents may have *limitations*?

By **learning**, we mean the process of an agent changing its behavior through experience to improve its ability to achieve a goal or accumulate long-term reward. Learning occurs through the agent's interaction with the environment: gaining percepts and rewards from the world and taking actions to affect it.

Learning is complicated by the existence of **other agents** also acting in the environment. We assume these other agents are *external* agents, that is, they are not under the control of the same agent programmer or designer. They are still likely to have goals of their own and to be learning to achieve those goals. As external agents, though, we can make few, if any, assumptions about their goals, algorithms, protocols, assumptions, or capabilities.

Complex domains are primarily domains with a very large or continuous set of possible configurations. In these domains, the pertinent dynamics of the environment depend on features that are either continuous or in their joint product are beyond enumeration. Often the complexity actually comes from the other agents in the domain. For example, in spatial domains the state of the environment usually includes the state or position of the agents themselves. So, the complexity of the domain inherently increases with the increase in the number of agents. These domains require the use of approximations by agents in order to act effectively in the domain.

Finally, **limitations** are anything that prevent agents from acting optimally. With our definition of complex domains, limitations are unavoidable. For example, approximation itself limits the agents from selecting optimal actions. Limitations also affect both our own agent as well as the other agents in the environment. Learning with limitations implies both accounting for the agents own inability to act optimally, as well as exploiting limited opponents or compensating for limited teammates.

1.2 Approach

This thesis question is examined in the context of the framework of stochastic games. Stochastic games were first studied extensively in the field of game theory, but can be viewed as a generalization of Markov decision processes (MDPs). MDPs have served as the foundation of much of the research in single-agent control learning. Stochastic games subsume both MDPs as well as the more well-known game theoretic model of matrix games. Like matrix games, stochastic games do not always have a well-defined notion of optimal behavior for a particular agent. The most common solution concept in these games is Nash equilibria, intuitively defined as a particular behavior for all the agents where each agent is acting optimally with respect to the other agents' behavior. All stochastic games have at least one Nash equilibria. But, equilibria are only sensible if all the agents are fully rational, optimal, and unlimited. Since our goal includes learning with limited agents, we do not make equilibria the explicit goal of learning. We will, though, use it as a tool to understand the multiagent learning problem.

We approach this thesis question by first examining the issue of multiagent learning in stochastic games when agents are not limited. We then consider how limitations affect the multiagent learning scenario. We use the techniques developed in the unlimited setting to address the more challenging problem of learning with limited agents. Our ultimate goal, though, of handling limited agents influences our approach to multiagent learning even in unlimited settings. Our approach is to develop techniques that are theoretically justifiable in analyzable and unlimited settings but scale well to practical and challenging problems.

An important factor in our approach is the strong emphasis on how learning algorithms perform in self-play. Our interests are in techniques that are general and applicable to many multiagent problems, as well as being robust to a variety of learning situations. Therefore we must acknowledge the fact that the other agents' designers may be equally as innovative, or well-read in the latest literature. Hence, it is important to consider situations where the other agents are using the same or similarly advanced techniques. Hence, our analysis, theoretical and empirical, will examine situations of self-play, while also seeking to examine situations beyond self-play.

The final important point of our approach is our consideration of both competitive and non-competitive multiagent settings. We seek to avoid making assumptions about the goals or rewards of the other agents, and do not expect a priori that their behavior is cooperative or adversarial. Our theoretical and empirical examinations will look at both competitive and non-competitive domains. This generality, however, will also prevent the work from completely addressing some specific multiagent environments where stricter assumptions can be made. For example, we do not focus on the interesting and challenging problem of a team of agents cooperating to globally maximize a known and common objective. In the team learning task, the issue is one of distributed computation and learning, as opposed to the issues of strategic decision making and learning, which is the emphasis of our work.

1.3 Contributions

The key contributions of this dissertation are four-fold.

Variable learning rate and the WoLF principle for rational and convergent learning. Rationality and convergence are two properties we introduce as desirable for a multiagent learning algorithm. Previous learning techniques do not simultaneously achieve both properties. We present the concept of a variable learning rate and the WoLF ("Win or Learn Fast") principle toward achieving these properties. We demonstrate that WoLF is both theoretically grounded and evaluate it empirically in a variety of stochastic games.

Analysis of restricted policy spaces and their effect on equilibria. We introduce a novel and general model of the effect of limitations on agent behavior. We use this model to analyze the resulting impact of limitations on the concept of Nash equilibrium. This result not only affects our own investigation but has far reaching consequences for research on multiagent learning, in general.

GraWoLF as a general-purpose scalable multiagent learning algorithm. GraWoLF combines the WoLF variable learning rate with policy gradient learning techniques. The algorithm directly addresses the thesis question of effective learning in complex domains with other limited agents. We introduce the algorithm and show compelling empirical results of its applications in complex and challenging problems. This includes a robot learning task where learning and evaluation was done on the robot platform itself. This is the first application of any explicitly designed multiagent reinforcement learning techniques to domains with intractable or continuous state spaces.

Play-based team strategy for adapting to an unknown opponent. CMDragons is a RoboCup small-size robot soccer team faced with the challenge of competing against a variety of completely unknown opponents. We developed and implemented a team strategic architecture built around the notion of plays as coordinated team plans, with the ability to adapt play selection during the course of a game. The CMDragons team represents the first use of online learning by a RoboCup robot team to autonomously alter its behavior during the course of a game, improving its behavior against its specific opponent. We demonstrate both the effectiveness of learning in the challenging multiagent domain of robot soccer. We also present this approach as an example of learning being effectively integrated into a fully implemented system.

1.4 Guide to the Thesis

Here we outline the chapters that follow.

Chapter 2 – A Framework for Multiagent Learning. In this chapter we present the general framework of stochastic games as a description of multiagent domains. We arrive at this framework by examining the two simpler frameworks: Markov decision processes for single-agent domains, and matrix games for state-less multiagent domains. We review the basics of these frameworks, exploring both results from reinforcement learning and game theory. This chapter forms the foundations upon which the rest of our work is built.

Chapter 3 – Categorization of Previous Work. We present a novel categorization of the related work addressing our thesis question of multiagent learning. We examine both game theoretic techniques for solving stochastic games, as well as previous algorithms for learning within the stochastic game framework, drawing connections between the two bodies of literature. We introduce the typology of equilibria learners and best-response learners as a useful classification of previous multiagent learning algorithms. This chapter also motivates our introduction of the WoLF principle for rational and convergent learning in the next chapter.

Chapter 4 – WoLF: Rational and Convergent Learning. We define two desirable properties for a learning algorithm in a multiagent setting: rationality and convergence. We show that the previous work discussed in Chapter 3 does not simultaneously achieve these properties. We then introduce the WoLF variable learning rate as a key contribution of this dissertation. We prove that WoLF can make a rational learning algorithm known not to converge, converge in a subclass of matrix games. We then demonstrate the effectiveness of the WoLF principle in a more practical form in a variety of stochastic games, with enumerable, but relatively large state spaces.

Chapter 5 – Analyzing Limitations. We examine limitations in the multiagent learning problem. We present a formal definition for the effect of limitations on agent behavior. We then use this formalization to investigate the effect of limitations on the concept of equilibria, an important concept for learning in stochastic games. We show that equilibria do not exist in general when agents are limited, even with very well-behaved limitations. We do, though, prove that equilibria do exist for a number of subclasses of games and limitations. We also demonstrate that a WoLF algorithm can effectively learn and converge to these equilibria when they exist.

Chapter 6 – GraWoLF: Learning with Limitations. This chapter introduces the GraWoLF algorithm, a general-purpose multiagent learning technique capable of learning in large games even in the presence of other limited agents. This technique combines policy gradient ascent techniques using a policy parameterization with the WoLF principle to learn in complex domains with other learning agents. We demonstrate the effectiveness of this technique in two interesting domains, one with an intractably large state space, and the other being a real robot task with a continuous state space.

Chapter 7 – Team Adaptation. We examine the problem of a team of agents adapting to a particular adversary. This problem has many similarities to our investigation of learning in stochastic games, but also has many differences. We describe our CMDragons small-size robot soccer team, which employed a play-based team architecture that adapted during a game to the current opponent.

Chapter 8 – Conclusion. We conclude this work with a review of our contributions along with a consideration of future work in the area of multiagent learning.

All readers should begin with Chapter 2, which provides the mathematical grounding as well as the foundational background critical to our addressing of the thesis question. Those interested in the WoLF or GraWoLF contributions should either continue to Chapter 3 to gain perspective on previous techniques for multiagent learning, or proceed to Chapter 4. Those interested in understanding the challenge limitations pose to the Nash equilibrium concept in stochastic games could skip these two chapters and go on to Chapter 5. Those interested in the application of WoLF to complex problems could proceed from Chapter 4 to Chapter 6. Chapter 6 refers to results in Chapter 5, but does not depend critically on ideas from that chapter. Chapter 7 is an independent result from these previous chapters and could be read at any time, although, the foundations of Chapter 2 may still be helpful to the reader.

Chapter 2

A Framework for Multiagent Learning

Frameworks are models of reality. As such, they are an important foundation for the generation and evaluation of new ideas. They establish the "rules of the game," crystallize the core issues, provide a common basis of study, make intrinsic assumptions visible, provide a general perspective on large classes of problems, help to categorize the variety of solutions, and allow comparison with other models of reality. It is with all of these reasons in mind that we begin this work by introducing a framework for multiagent learning.

Specifically, we consider the framework of stochastic games. Stochastic games are best viewed as the synthesis of two simpler frameworks: Markov decision processes and matrix games. See Figure 2.1 for an overview of this synthesis. Markov decision processes have been prominently explored in the field of reinforcement learning, while matrix games are the foundational concept of the field of game theory. Markov decision processes are a single agent, multiple state model. Matrix games, on the other hand, are a multiagent, but single state model. Stochastic games can be seen as unifying and subsuming these two frameworks, defining *a multiagent, multiple state framework*. Since stochastic games share concepts with these two separate frameworks, it is useful to consider them each independently. The separate presentation also helps to crystallize the core issues in stochastic games as they are distinguished from these simpler models. This chapter, in addition, outlines the key concepts and results that we make use of later in the work.

We begin in Section 2.1 with a very brief overview of the general model of an agent and agent learning, which underlies all of the frameworks we discuss. In Section 2.2 we focus on Markov decision processes. We provide an overview of the model, relevant solution concepts, and basic theoretical foundations. We then explore some standard learning algorithms as well as some extensions that use approximation to apply learning to problems with intractably sized state spaces. This work is very relevant to our consideration of agent limitations in Chapters 5 and 6. In Section 2.3 we present the framework of matrix games. We discuss the various solution concepts, a classification of games, and basic theory. Finally, in Section 2.4 we present the stochastic game framework. In its entirety this Chapter overviews the necessary foundations upon which the rest of this work is built.

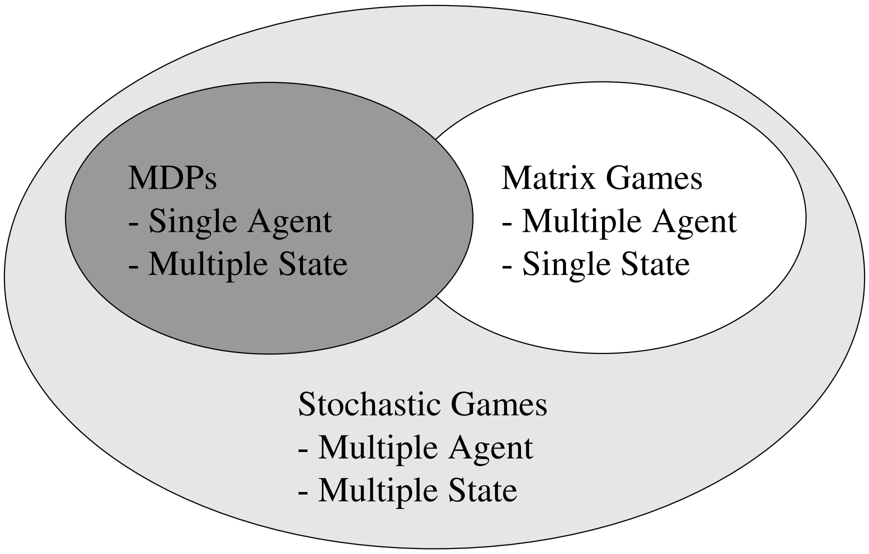


Figure 2.1: Matrix games, Markov decision processes, and stochastic games.

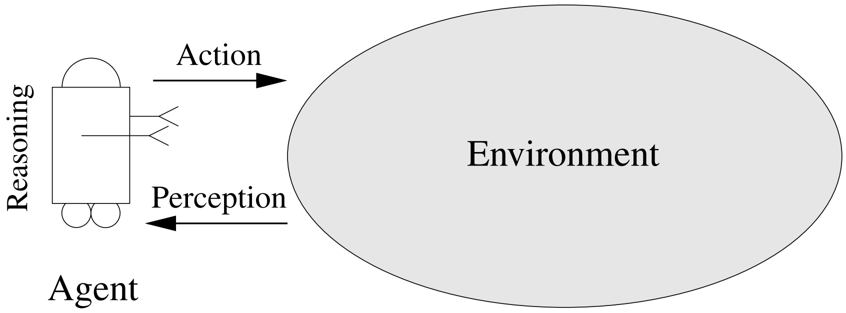


Figure 2.2: Agent framework: agents are distinguished from their environment and are composed of perception, reasoning, and action.

2.1 Agent Framework

All agents have three key components: perception, reasoning, and action. These three components operate within the context of some environment as shown graphically in Figure 2.2. The percepts an agent receives depends on the environment, and the actions the agent performs, in turn, affects the environment. The frameworks we describe in this chapter define a specific structure for the environment: how it is affected by the agent's actions, how it affects the agent's percepts, and whether other agents are involved. With general but careful assumptions about the environment, agents can effectively reason about appropriate actions to select.

In learning agents, which are the focus of this work, there are two additional factors. First, the details of the environment are initially unknown. The agent only receives information about the environment through its interaction, that is, by selecting actions and observing their effects through its perceptual inputs. Second, there is an additional input signal which is the agent's reward. This reward depends on the environment and the agent's actions. The reasoning portion of the agent is

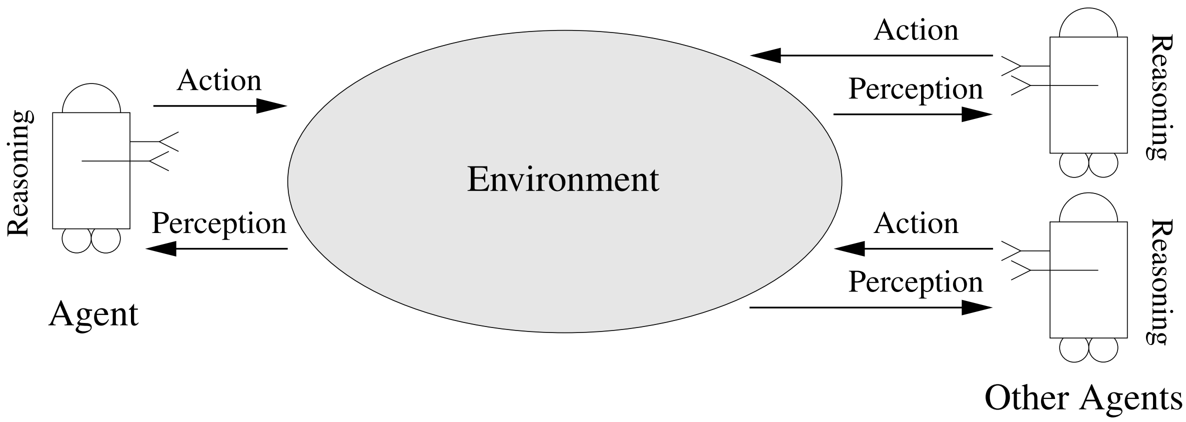


Figure 2.3: Multiagent framework: multiple agents all distinguished from their environment and composed of perception, reasoning, and action.

a learning process that repeatedly interacts with the environment with the goal of maximizing the rewards it receives over time.

Another common consideration in learning frameworks is the issue of observability. What is the nature of the agent's perception? A common assumption and one that we focus on in this work is that the environment is fully observable. This means that an agent's perception consists of the entire relevant state of the environment. Some models relax this assumption by allowing the agent to receive *observations* that are contingent upon, but do not uniquely determine, the full state of the environment. Although, this work focuses on fully observable environments, in our examination of agent limitations in Section 5 we consider the affect of observability on agent behavior. We also, in Section 6, show results of learning in a robot problem that violates the fully observable assumption.

This work is focused on learning *in the presence of other agents*. Figure 2.3 depicts this graphically. Instead of a single agent perceiving, reasoning, and acting in an environment, there are multiple complete agents. These agents also receive perceptions, reason, and act on the environment. Additionally, they may be learning agents as well, adapting their actions to maximize their own reward signal over time.

We now look at three formalizations of the agent and multiagent frameworks: Markov decision processes, matrix games, and stochastic games. Markov decision processes correspond to the basic agent framework of Figure 2.2. Matrix games consider multiple agents in a single-state environment, that is rewards to the agents depend solely on the agents' actions. Stochastic games correspond to the full multiagent framework depicted in Figure 2.3. Although, ultimately we focus on the subsuming model of stochastic games, it is very useful to understand the simpler models (see again Figure 2.1). For completeness, we briefly mention the category that is the intersection, rather than the union, of matrix games and Markov decision processes. These are single state, single agent models and are referred to as *k-armed bandit problems*. They amount to the problem of selecting from which of k slot machines to receive a stochastic payoff. These problems have many similarities to Markov decision processes and for further information we recommend Kaelbling, Littman, and Moore's reinforcement learning survey (1996). We now explore the models of Markov decision processes, matrix games, and finally stochastic games, in turn.

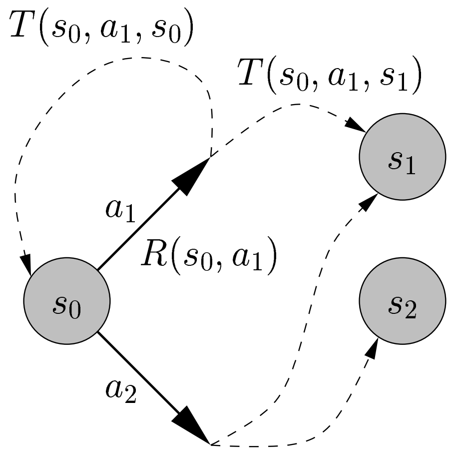


Figure 2.4: Graphical depiction of the Markov decision process (MDP) framework. Not all actions and transitions are shown.

2.2 Markov Decision Processes

Markov decision processes are the foundation for much of the research in single agent control learning. A *Markov decision process* (MDP) (Bellman, 1957; Sutton & Barto, 1998) is a tuple, $(\mathcal{S}, \mathcal{A}, T, R)$, where,

- $\mathcal{S}$ is the set of states,
- $\mathcal{A}$ is the set of actions,
- T is a transition function, $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, such that,

$$\forall s \in \mathcal{S}\ \forall a \in \mathcal{A} \qquad \sum_{s' \in \mathcal{S}} T(s, a, s') = 1,$$

- and R is a reward function, $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

This definition is depicted graphically in Figure 2.4. The transition function defines a probability distribution over next states as a function of the current state and the agent's action. The reward function defines the reward received when selecting an action from the given state. The agent interacts with an MDP by alternating between perception and action. the agent observes the state at time t, s^t, and selects an action a^t. The agent receives the reward $r^t = R(s^t, a^t)$, and the agent observes the new state, s^{t+1} drawn from the probability distribution specified by $T(s^t, a^t, s^{t+1})$. We use the notation,

$$s^0, a^0, r^0, \quad s^1, a^1, r^1, \quad \ldots \quad s^t, a^t, r^t, \quad \ldots$$

to refer to a single trace of execution for a given MDP and selection of actions.

Notice that this is a single agent formalization of the agent framework. The agent's perceptions are the current state of the environment from the set $\mathcal{S}$. The agent's reasoning process, or learning algorithm, processes the observed state and reward and is responsible for selecting an action from the set $\mathcal{A}$. This closes the loop of agent perception, reasoning, and action.

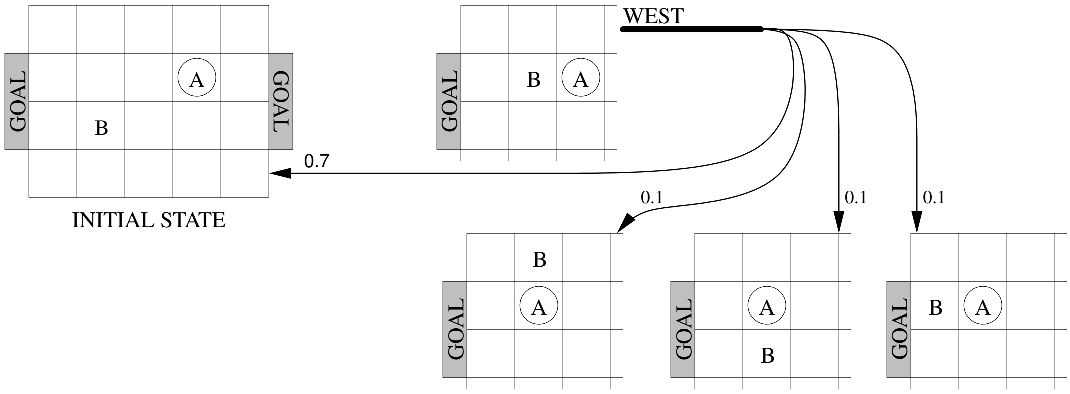


Figure 2.5: Single player grid soccer where agent **B** selects its actions randomly. The initial state is shown on the left and an example of the transitions and associated probabilities are given for a particular state and action on the right. Notice that fifty percent of the time **A**'s action is executed first, causing it to lose the ball and the game to reset to the initial state. In addition, if **B** selects **H** or **E** it does not move and so **A** still loses the ball and returns to the initial state. The other outcomes all have equal probability.

2.2.1 An Example: Single Player Grid Soccer

Consider the soccer-inspired grid domain with twenty cells and two agents, **A** and **B**. An example state is shown in Figure 2.5. In the single player definition of the domain, the agent, marked by **A**, is in possession of the ball and is trying to score in the left goal. The state is defined by the non-identical positions of the agent and a defender, marked **B**. This defines a state space, $\mathcal{S}$, where $|\mathcal{S}| = 20 \times (20 - 1) = 380$. The actions are movements in the four compass directions: **N**, **S**, **E**, **W**, and the hold action, **H**. The next state is determined by the selected action of the agent and a randomly selected action of the defender. The defender's action is selected from the same set of four compass directions and the hold action, all with equal probability. The agent's and defender's actions are then executed in random order, where an action to move into the other agent's location is handled specially. If the defender tries to move into the agent's location it does not move. If the attacker moves into the defender's location the state is reset to the initial configuration which is the state shown in Figure 2.5. A reward of 1 is received when the agent enters the goal, i.e., selects the **E** action from the two leftmost, central grid locations. The state, in this case, is then reset to the initial configuration. For all other cases, the reward is zero. These rules formally define a stochastic transition function, T, and reward function, R. An example of the transitions and associated probabilities for a single state and action are shown in Figure 2.5.

2.2.2 Solution Concepts

The goal of a learning agent in an MDP is to learn a policy so as to maximize its long-term reward. The transition function T and reward function R are not known in advance, and only samples are received as the agent selects actions in the environment. A policy, π, is a mapping that defines the probability of selecting an action from a particular state. Formally, $\pi \in \mathcal{S} \times \mathcal{A} \to [0,1]$, where,

$$\forall s \in \mathcal{S} \qquad \sum_{a \in \mathcal{A}} \pi(s,a) = 1.$$

When learning in MDPs, the space of policies considered is often restricted to only deterministic policies, where $\pi(s,a) \in \{0,1\}$. Since stochastic policies are an important feature of some recent work in MDPs, which we overview in Section 2.2.3, as well as our work in stochastic games, we focus on this more general class.

The goal of maximizing long-term reward can be formulated in a number of different ways. We concentrate on the *discounted reward* formulation with some of our results also applying to the *average reward* formulation.

Discounted Reward

In the discounted reward formulation, immediate reward is preferred over future reward. Specifically, the value of a policy, π, at state, s, with a discount factor, $\gamma \in [0,1)$, is,

$$V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t E\{r^t | s^0 = s, \pi\}, \tag{2.1}$$

where $E\{r^t | s^0 = s, \pi\}$ is the expected reward received at time t given the initial state is s and the agent follows the policy π. V^π is called the policy's state value function. This formulation is similar to the economic principle of interest and investment, where utility now is traded against larger future utility. It can also be understood as describing the possibility that the process itself will terminate after any step with probability γ, after which no additional reward can be accumulated. Another reason for considering discounted reward is that it simplifies the mathematics.

Using this formulation, the goal of the agent is to learn an optimal policy, π^*, that maximizes discounted future reward at all states,

$$\forall \pi \ \forall s \in \mathcal{S} \qquad V^{\pi^*}(s) \geq V^\pi(s).$$

A policy's value function is known to be the unique solution to a recursive equation called the Bellman equation (Howard, 1960),

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s') \right). \tag{2.2}$$

The value function for all optimal policies is identical, denoted V^*, and must satisfy an even stronger relation called the Bellman optimality equation,

$$V^*(s) = \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^*(s') \right). \tag{2.3}$$

Notice that this must be the case since a suboptimal policy can be improved by changing the policy to select the action that maximizes the parenthesized expression on the right-hand side of

Equation 2.2. In essence the Bellman optimality equation requires that the policy's value already maximizes this expression. This fact of policy improvement is the basis for a number of learning algorithms, two of which are examined in the next section. In addition, the optimal value function also defines an optimal policy. Specifically, the policy that deterministically selects the action that maximizes the right-hand side of Equation 2.3 is an optimal policy.

Another useful concept is a policy's state-action value function, also known as its Q-values. This defines the expected discounted reward of choosing a particular action from a particular state and then following the policy π. From Equation 2.2 we can write this as,

$$Q^\pi(s,a) \quad = \quad R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s'). \tag{2.4}$$

Similarly, the optimal Q-values, denoted Q^*, satisfies,

$$\begin{aligned} Q^*(s,a) \quad &= \quad R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^*(s') \\ &= \quad R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') \max_{a \in \mathcal{A}} Q^*(s',a). \end{aligned} \tag{2.5}$$

Notice that the policy improvement rule from above now corresponds to changing the policy to select the action that maximizes its Q-value for the current state. By considering Q-values we can also completely define the space of optimal policies. An optimal policy is any policy that from state s has all of its probability assigned to actions that maximize $Q^*(s,a)$.

Average Reward

An alternative formulation is that of *average reward*. Although we focus on discounted reward, some of our theoretical results in Chapter 5 also apply to the average reward formulation. In this formulation the value of a policy is its long-term expected reward per time step. Mathematically, the value of a policy, π, at state, s, is,

$$V^\pi(s) = \lim_{T \to \infty} \sum_{t=0}^{T} \frac{1}{T} E\{r^t | s^0 = s, \pi\}. \tag{2.6}$$

A common assumption, which usually accompanies examinations of this reward formulation, is that the MDP is *unichain*. An MDP is unichain if and only if, for all policies, there exists an *ergodic* set of states (i.e., any state in the set can be reached with non-zero probability from any other state in the set), and all states outside this set are *transient* (i.e., after some finite point in time it will never be visited again). This assumption forces the value of a policy to be independent of the initial state. From any initial state, the policy is guaranteed to end up in the ergodic class, and any state in the ergodic class must have the same average reward for a given policy.

2.2.3 Reinforcement Learning

Reinforcement learning (RL) is both a field and a class of algorithms concerned with the problem of learning in MDPs. We build extensively on ideas and algorithms from RL and so we briefly describe some basic algorithms here. We also examine a few recent developments for applying learning to very large MDPs. This will provide both a motivation and a foundation for our consideration of limitations and learning in later chapters.

The crux of most RL techniques is the learning of a value function. As we noted in Section 2.2.2, the optimal value function defines an optimal policy. Also, a policy's value function can be used to improve that policy. The traditional goal of RL algorithms is to compute the optimal value function for an unknown and arbitrary MDP. The algorithms select actions and receive observations of the reward and the next state reached according to the reward function and transition probabilities. We look at two different algorithms that use these inputs: Q-Learning and Sarsa(λ). Both of these techniques make use of a similar idea for estimating a value function called *temporal differencing* (Sutton & Barto, 1998).

Temporal differencing is the idea of turning Equations 2.4 or 2.5 into assignment operators. Since we want a value function that satisfies one of those equations, we can simply force the equation to hold by changing the left-hand side to equal the right. Roughly speaking, Q-learning will do this for Equation 2.5 and Sarsa(λ) will do this for Equation 2.4. In both cases, though, since T is not known, they will use a stochastic estimation procedure based on samples from T through experience with the environment. This latter idea is that of temporal differencing. We will generally use the name temporal differencing to refer to the updating of a value function based on the current estimate of the value of future states, whether estimation is used or not. In Chapter 3 we will see how the idea of temporal differencing will help unify some of the existing game theory and reinforcement learning algorithms for stochastic games.

Q-Learning

One of the most basic RL algorithms is Q-learning (Watkins, 1989). The algorithm learns the optimal state-action value function, Q^*, which also defines an optimal policy. In its simplest form, the learner maintains a table containing its current estimates of $Q^*(s, a)$. It observes the current state s and selects the action a that maximizes $Q(s, a)$ with some exploration strategy. Upon receiving a reward, r, and observing the next state, s', it updates its table of Q-values according to the following rule,

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a) \right), \tag{2.7}$$

where $\alpha \in (0, 1)$ is the learning rate.

Under certain conditions, when Q is updated according to this rule, it is guaranteed to converge to the optimal Q^*. The specific conditions are (i) every state-action pair is updated an infinite number of times. And, (ii) α is appropriately decayed over time; specifically, if α_t is the learning rate used at time t, then it must satisfy,

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty.$$

Condition (ii) is a standard requirement for stochastic approximation. Condition (i) emphasizes the importance of the exploration strategy. Throughout this work we assume that suitable exploration policies are used (see (Singh, Jaakkola, Littman, & Szepesvári, 2000a)). One such exploration policy is ϵ-greedy. With this policy, the action currently estimated to be optimal is selected with probability $1 - \epsilon$, otherwise, with probability ϵ, a random action is chosen. If $\epsilon > 0$ approaches zero over time, then this satisfies Condition (i) with the agent's actions approaching the optimal actions.

Sarsa(λ)

Sarsa (Rummery & Niranjan, 1994; Sutton & Barto, 1998) is another RL algorithm that learns a state-action value function. However, instead of learning Q^* it learns Q^π for the current policy, but then selects actions so as to maximize its current Q estimates. As we noted above, this change of policy, necessarily improves the values of the policy. In its simplest form, it is very similar to Q-learning. It maintains a table of Q-values. It observes the current state, s, and selects action, $a = \operatorname{argmax}_a Q(s, a)$, with some exploration strategy. Upon receiving a reward, r, observing the next state, s', and selecting action, a', by the same method, it updates its table according to,

$$Q(s, a) \leftarrow Q(s, a) + \alpha\Big(r + \gamma Q(s', a') - Q(s, a)\Big), \tag{2.8}$$

where α is, again, the learning rate. Under the same conditions as Q-learning, this rule is also guaranteed to converge to Q^* (Littman & Szepesvári, 1996).

Since the Q-table now estimates the value of the current policy, Q^π, there is an additional optimization that can be used to speed this estimation. It is commonly called an eligibility trace. One way of viewing eligibility traces is that they propagate errors in Q value estimates more quickly through the table. An eligibility trace, $e(s, a)$, defines the state-action pair's responsibility for the current error in Q-values. After observing state, s, selecting action, a, receiving reward r, observing s' and selecting a', it updates its trace and Q-table according to,

$$\forall \hat{s} \in \mathcal{S}\ \hat{a} \in \mathcal{A} \qquad e(\hat{s}, \hat{a}) \leftarrow \lambda\gamma e(\hat{s}, \hat{a}) + \begin{cases} 1 & \text{if } \hat{s} = s \text{ and } \hat{a} = a \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

$$Q(\hat{s}, \hat{a}) \leftarrow Q(\hat{s}, \hat{a}) + \alpha e(\hat{s}, \hat{a})\Big(r + \gamma Q(s', a') - Q(s, a)\Big), \tag{2.10}$$

where $\lambda \in [0, 1]$ is a parameter controlling the amount of error propagated. If λ is zero, notice that this becomes the basic Sarsa rule. If λ is one, the rule becomes similar to a Monte-Carlo estimation of the value of the policy. In practice, a value of λ strictly between zero and one can often greatly speed learning. The best choice of λ, though, varies from problem to problem.

Approximation

The above learning algorithms require a table of state-action values. Many problems, though, have an enormous or continuous state space making state enumeration impossible. Large state spaces are problematic due to both the required memory of storing a table of values and the vast amounts of training experience needed to experience the transitions. This critical problem has received vast treatment in the reinforcement learning literature (for a summary of this work see Chapter 8 in (Sutton & Barto, 1998) or Section 6 of (Kaelbling et al., 1996)).

The problem of scalability is also one of the critical problems that motivates the emphasis of this thesis on learning when agents have limitations. The proposed solutions to this problem involve approximation and generalization. Approximation is needed to reduce the number of parameters that need to be optimized. This is the usual tradeoff of learning suboptimal policies in exchange for more efficient learning. Generalization is needed so that training experience can be used more efficiently. We present some of the specifics of these techniques and their effect on the learning problem, specifically from a multiagent perspective, in Chapter 5. We also describe in detail in Chapter 6 one particular technique, policy gradient ascent (Williams & Baird, 1993; Sutton, McAllester, Singh, & Mansour, 2000; Baxter & Bartlett, 2000), and use it in constructing a scalable multiagent learning algorithm.

2.2.4 The Markov Assumption

Markov decision processes are called such since they satisfy the *Markov Assumption*. Basically, this requires that the next state and reward to the agent is fully specified by the current state and agent's action. We can state this property formally.

Definition 1 *A decision process is* Markovian *if and only if, the sequence of states ($s^t \in \mathcal{S}$), actions ($a^t \in \mathcal{A}$), and rewards ($r^t \in \mathbb{R}$), satisfies*

$$Pr\left\{s^t = s, r^t = r | s^{t-1}, a^{t-1}, \ldots, s^0, a^0\right\} = Pr\left\{s^t = s, r^t = r | s^{t-1}, a^{t-1}\right\},$$

that is, if the next state depends only on the previous state and agent's action, and not on the history of states and actions.

An agent's selection of actions is Markovian *if and only if,*

$$Pr\left\{a^t = a | s^t, s^{t-1}, a^{t-1}, \ldots, s^0, a^0\right\} = Pr\left\{a^t = a | s^t\right\},$$

that is, if the agent's next action depends only on the current state. Sometimes we refer to a Markovian policy as stationary *since the policy does not change with respect to time.*

The MDP framework and our definition of a policy, by construction, are Markovian. When we explore the multiagent framework of stochastic games later in this chapter, this property comes into question.

2.3 Matrix Games

Matrix games were first examined in the field of game theory to specifically model strategic interactions of many decision makers. A *matrix game* or *strategic game* (von Neumann & Morgenstern, 1944; Osborne & Rubinstein, 1994) is a tuple $(n, \mathcal{A}_{1\ldots n}, R_{1\ldots n})$, where,

- n is the number of players,
- $\mathcal{A}_i$ is the set of actions available to player i (and $\mathcal{A}$ is the joint action space $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n$),
- and R_i is player i's payoff function, $\mathcal{A} \rightarrow \mathbb{R}$.

The players select actions from their available set and receive a payoff that depends on *all* the players' actions. These are often called matrix games, since the R_i functions can be written as n-dimensional matrices. The actions then correspond to specifying the value of a particular dimension, and the joint actions correspond to particular entries in the reward matrices. Since we are interested in learning, we will focus on agents repeatedly playing the same matrix game. In game theory, this is called a *repeated game*. Since this thesis focuses on learning and, therefore, is not interested in the one-shot playing of a matrix game, we will use both terms to mean agents repeatedly playing and learning a matrix game.

As in MDPs, the agents in a repeated game have an explicit action set. Unlike MDPs, though, the environment has no state and so the agents' only perception is the actions of the other agents, or maybe just their own reward. The agents must reason or learn through experience how to select their action to maximize their observed reward.

2.3.1 Examples

Table 2.1 contains a number of example matrix games. The games (a)–(e) are two player games, i.e. $n = 2$, and (f) is a three player game, i.e. $n = 3$. The table only specifies the games' R_i matrices and not the individual players' action sets, which unless otherwise defined are just assumed to be indices into the reward matrices. Also, in two player games, the player specifying the row index is always player 1, and the column player always player 2.

Matching pennies (a) and rock-paper-scissors (b) are two children's games that can be captured as matrix games. In matching pennies the players' actions are "Heads" (**H**) and "Tails" (**T**). Player one takes a dollar from player two if their choices match, and gives a dollar if their choices differ. Rock-paper-scissors is a similar game with each player having three actions: "Rock" (**R**), "Paper" (**P**), and "Scissors" (**S**). The rules specify that "Rock" loses to "Paper", "Paper" loses to "Scissors", and "Scissors" loses to "Rock". Otherwise the game is a tie. The winner, if there is one, takes a dollar form the loser.

The coordination game (c) and Bach–or–Stravinsky (d) are completely different games. In the coordination game, both players simply desire to agree on their action choice, but have no preferences between them. In Bach–or–Stravinsky two friends are deciding independently what concert to attend. Player 1 prefers Bach, while player 2 prefers Stravinsky, but both prefer to be together over being apart.

In Colonel Blotto (e) (Gintis, 2000), two generals are allocating armies between two battlefields. The first general has four armies, while the second only has three. The general with the most armies at a battlefield gets one victory point, plus the number of opponent armies defeated. Ties give no points to either general. Each action, i.e., row or column, corresponds to the possible allocation of its armies to the two battlefields.

Finally, three player matching pennies (f) adds an additional player to the matching pennies game. Player 1 gets one dollar for matching player 2. Player 2 gets one dollar for matching player 3, and player 3 gets one dollar for *not* matching player 1. The three dimensional tensor shown in the Table is somewhat challenging to follow. Player 1 selects the row index, player 2 selects the column index, and player 3 selects the left or right column of matrices.

$$R_1 = \begin{array}{c} \\ \mathbf{H} \\ \mathbf{T} \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{H} & \mathbf{T} \end{array} \\ \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \end{array}$$

$$R_2 = \begin{array}{c} \mathbf{H} \\ \mathbf{T} \end{array} \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

(a) Matching Pennies

$$R_1 = \begin{array}{c} \\ \mathbf{R} \\ \mathbf{P} \\ \mathbf{S} \end{array} \begin{array}{c} \begin{array}{ccc} \mathbf{R} & \mathbf{P} & \mathbf{S} \end{array} \\ \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \end{array}$$

$$R_2 = \begin{array}{c} \mathbf{R} \\ \mathbf{P} \\ \mathbf{S} \end{array} \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}$$

(b) Rock–Paper–Scissors

$$R_1 = \begin{array}{c} \\ \mathbf{H} \\ \mathbf{T} \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{H} & \mathbf{T} \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$$

$$R_2 = \begin{array}{c} \mathbf{H} \\ \mathbf{T} \end{array} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

(c) Coordination Game

$$R_1 = \begin{array}{c} \\ \mathbf{B} \\ \mathbf{S} \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{B} & \mathbf{S} \end{array} \\ \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$$

$$R_2 = \begin{array}{c} \mathbf{B} \\ \mathbf{S} \end{array} \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

(d) Bach–or–Stravinsky

$$R_1 = \begin{array}{c} \\ \mathbf{4\text{-}0} \\ \mathbf{3\text{-}1} \\ \mathbf{2\text{-}2} \\ \mathbf{1\text{-}3} \\ \mathbf{0\text{-}4} \end{array} \begin{array}{c} \begin{array}{cccc} \mathbf{3\text{-}0} & \mathbf{2\text{-}1} & \mathbf{1\text{-}2} & \mathbf{0\text{-}3} \end{array} \\ \begin{pmatrix} 4 & 2 & 1 & 0 \\ 1 & 3 & 0 & -1 \\ -2 & 2 & 2 & -2 \\ -1 & 0 & 3 & 1 \\ 0 & 1 & 2 & 4 \end{pmatrix} \end{array}$$

$$R_2 = \begin{array}{c} \mathbf{4\text{-}0} \\ \mathbf{3\text{-}1} \\ \mathbf{2\text{-}2} \\ \mathbf{1\text{-}3} \\ \mathbf{0\text{-}4} \end{array} \begin{pmatrix} -4 & -2 & -1 & 0 \\ -1 & -3 & 0 & 1 \\ 2 & -2 & -2 & 2 \\ 1 & 0 & -3 & -1 \\ 0 & -1 & -2 & -4 \end{pmatrix}$$

(e) Colonel Blotto

$$R_1(\langle\cdot,\cdot,\mathbf{H}\rangle) = \begin{array}{c} \\ \mathbf{H} \\ \mathbf{T} \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{H} & \mathbf{T} \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \qquad R_1(\langle\cdot,\cdot,\mathbf{T}\rangle) = \begin{array}{c} \\ \mathbf{H} \\ \mathbf{T} \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{H} & \mathbf{T} \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$$

$$R_2(\langle\cdot,\cdot,\mathbf{H}\rangle) = \begin{array}{c} \mathbf{H} \\ \mathbf{T} \end{array} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \qquad R_2(\langle\cdot,\cdot,\mathbf{T}\rangle) = \begin{array}{c} \mathbf{H} \\ \mathbf{T} \end{array} \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

$$R_3(\langle\cdot,\cdot,\mathbf{H}\rangle) = \begin{array}{c} \mathbf{H} \\ \mathbf{T} \end{array} \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \qquad R_3(\langle\cdot,\cdot,\mathbf{T}\rangle) = \begin{array}{c} \mathbf{H} \\ \mathbf{T} \end{array} \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$$

(f) Three Player Matching Pennies

Table 2.1: Examples of matrix games.

2.3.2 Solution Concepts

The goal of a learning agent in a matrix game is to learn a strategy that maximizes its reward. A *pure strategy* is one that deterministically selects a single action. However, pure strategies in matrix games can often be exploited. For example, consider matching pennies in Table 2.1(a). If the player plays either action deterministically then the other player can guarantee to win a dollar by playing the appropriate action. A *mixed strategy* for player i, $\sigma_i(a_i \in \mathcal{A}_i)$, specifies a probability distribution over actions. These are the strategies we focus on. We use σ_i to refer to one particular player's strategy, and σ to refer to a strategy for all of the players, called the *joint strategy*. We extend the reward function, R_i, to be defined over mixed strategies as well,

$$R_i(\sigma) = \sum_{a \in \mathcal{A}} R_i(a) \Pi_{i=1}^{n} \sigma_i(a_i).$$

We also use σ_{-i} to refer to a joint strategy for all of the players except player i. Finally, we will use the notation $\langle \sigma_i, \sigma_{-i} \rangle$ to refer to the joint strategy where player i follows σ_i while the other players follow their policy from σ_{-i}.

In MDPs a solution is defined as the policy with the highest value according to some reward formulation, such as discounted reward. In matrix games, no single optimal strategy exists. A strategy can only be evaluated if the other players' strategies are known. This can be illustrated in the matching pennies game (Table 2.1 (a)). In this game, if player 2 is going to play **H**, then player 1's optimal strategy is to play **H**, but if player 2 is going to play **T**, then player 1's optimal strategy is to play **T**. In this game, there is no strategy, pure or mixed, that is optimal independent of the opponent. What does exist is an opponent-*dependent* solution, or set of solutions. This is called a *best-response*.

Definition 2 *For a matrix game, the* best-response function *for player i, $\mathrm{BR}_i(\sigma_{-i})$, is the set of all strategies that are optimal given the other player(s) play the joint strategy σ_{-i}. Formally, $\sigma_i^* \in BR_I(\sigma_{-i})$ if and only if,*

$$\forall \sigma_i \in PD(A_i) \qquad R_i(\langle \sigma_i^*, \sigma_{-i} \rangle) \geq R_i(\langle \sigma_i, \sigma_{-i} \rangle),$$

where $PD(A_i)$ is the set of all probability distributions over the set A_i, i.e., the set of all mixed strategies for player i.

The major advancement that has driven much of the development of matrix games and game theory is the notion of a best-response equilibrium or *Nash equilibrium* (Nash, Jr., 1950).

Definition 3 *A* Nash equilibrium *is a collection of strategies for all players, σ_i, with*

$$\sigma_i \in \mathrm{BR}_i(\sigma_{-i}).$$

So, no player can do better by changing strategies given that the other players continue to follow the equilibrium strategy.

What makes the notion of equilibrium compelling is that all matrix games have a Nash equilibrium, although there may be more than one.

2.3.3 Types of Matrix Games

Matrix games can be usefully classified according to the structure of their payoff functions. Two common classes of games are *team games* and *zero-sum games*. In team games (e.g., coordination game in Table 2.1(c)) all agents have the same payoff function, so a joint action in the best interest of one agent is in the best interest of all the agents. In zero-sum games, there are two agents, and one's reward is always the negative of the other (i.e., $\forall a \in \mathcal{A} R_1(a) + R_2(a) = 0$).[1] The games (a), (b), and (e) in Table 2.1(a) are examples of such a game. The term *general-sum games* is used to refer to all types of games, although often means only non-zero-sum games. Notice that games (d) and (f) from Table 2.1 are neither team games nor zero-sum, despite looking very similar to a team game and zero-sum game, respectively.

One appealing feature of zero-sum games is that they contain a unique Nash equilibrium.[2] This equilibria corresponds to the game's minimax solution, or the mixed strategy that maximizes the worst-case expected reward. This equilibrium can be found as the solution to a linear program. The linear program has $|\mathcal{A}_1|$ parameters, one for each action, $\sigma(a_1 \in \mathcal{A}_1)$. These parameters are the player's probability distribution over actions, and so the linear program is,

$$\begin{array}{ll} \text{Maximize:} & \min_{a_2 \in \mathcal{A}_2} \sum_{a_1 \in \mathcal{A}_1} \sigma(a_1) R_1(\langle a_1, a_2 \rangle) \\ \text{Subject to:} & \sum_{a_1 \in \mathcal{A}_2} \sigma(a_1) = 1 \\ & \sigma(a_1) \geq 0 \quad \forall a_1 \in \mathcal{A}_1. \end{array}$$

The solution to this linear program is player 1's equilibrium strategy. Player 2's strategy could be similarly solved. In both matching pennies and rock-paper-scissors there is a unique equilibrium corresponding to both players selecting their actions with equal probability. In Colonel Blotto the equilibrium is also unique but is more complex:

$$\sigma_1 = \left\langle \frac{4}{9}, 0, \frac{1}{9}, 0, \frac{4}{9} \right\rangle \qquad \sigma_2 = \left\langle \frac{1}{18}, \frac{4}{9}, \frac{4}{9}, \frac{1}{18} \right\rangle.$$

For zero-sum games, an equilibrium can be computed efficiently as the solution to the above linear program. However, finding equilibria in two-player general-sum games requires a more difficult quadratic programming solution (Mangasarian & Stone, 1964). Beyond two-players things are even more difficult. McKelvey and McLennan (1996) survey a variety of techniques for computing equilibria in matrix games, including n-player general-sum matrix games.

This is the general case, though. In team games, where all players receive identical rewards, the joint action that maximizes one player's reward maximizes all players' rewards, and is therefore a Nash equilibrium. For example, in the coordination game, both players selecting the same action are two equilibria. In Bach–or–Stravinsky, although not strictly a team game, these same joint

[1] In the class of zero-sum games we also include all *constant-sum games*, since this constant can be subtracted from all the rewards turning the game into a zero-sum game. These games have identical properties to their zero-sum counterparts.

[2] There can actually be multiple equilibria, but they all have equal payoffs and are interchangeable (Osborne & Rubinstein, 1994). That is, if $\langle \sigma_1, \sigma_2 \rangle$ and $\langle \sigma_1', \sigma_2' \rangle$ are Nash equilibria, then $\langle \sigma_1, \sigma_2' \rangle$ and $\langle \sigma_1', \sigma_2 \rangle$ are also Nash equilibria, and $\forall i \; R_i(\langle \sigma_1, \sigma_2 \rangle) = R_i(\langle \sigma_1', \sigma_2' \rangle)$.

strategies are again equilibria, although each player would prefer one equilibrium over the other. In the final game, three player matching pennies, although it is not strictly zero-sum, the minimax optimal solution where all players select actions with equal probability is the unique equilibrium.

The formal complexity of constructing a Nash equilibrium for a general matrix game is an open problem. However, a number of related questions, such as identifying whether equilibria with certain properties exist, is in fact known to be NP-Hard (Gilboa & Zemel, 1989; Conitzer & Sandholm, 2003b). These results question the possible general efficiency of any equilibrium learning algorithms, both for matrix games and for stochastic games. From a practical perspective, this could limit the rate of convergence of equilibrium learning algorithms in their worst-case. This fact, though, will not come into play in our analysis.

2.4 Stochastic Games

Stochastic games are a superset of MDPs and matrix games, including both multiple agents and multiple states (see Figure 2.1). They were first examined in the field of game theory and now more recently in the field of reinforcement learning. A *stochastic game* (Shapley, 1953; Filar & Vrieze, 1997) is a tuple $(n, \mathcal{S}, \mathcal{A}_{1\ldots n}, T, R_{1\ldots n})$, where,

- n is the number of players,
- $\mathcal{S}$ is the set of states,
- $\mathcal{A}_i$ is the set of actions available to player i (and $\mathcal{A}$ is the joint action space $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n$),
- T is the transition function, $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, such that[3],

$$\forall s \in \mathcal{S}\ \forall a \in \mathcal{A} \qquad \sum_{s' \in \mathcal{S}} T(s, a, s') = 1,$$

- and R_i is the reward function for the ith agent, $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

This is shown graphically in Figure 2.6. Essentially, stochastic games can be thought of as an extension of MDPs to multiple agents, and have occasionally been referred to as multiagent MDPs (Boutilier, 1996) and competitive MDPs (Filar & Vrieze, 1997). Instead of a single agent, there are multiple agents whose *joint action* determines the next state and rewards to the agents. Also, notice that each agent has its own independent reward function.

Stochastic games can equally be thought of as an extension of the concept of matrix games to multiple states. Each stochastic game has a matrix game associated with each state. The immediate payoffs at a particular state are determined by the matrix entries $R_i(s, a)$. After selecting actions and receiving their rewards from the matrix game, the players are transitioned to another state and associated matrix game, which is determined by their joint action. Therefore, stochastic games contain as subsets of the framework: MDPs (when $n = 1$) and repeated games (when $|S| = 1$).

[3]If $\mathcal{S}$ is continuous then the following summation, along with other summations over $\mathcal{S}$ in this chapter, should be replaced by an integral. Likewise, if $\mathcal{A}_i$ is continuous then the use of summations over action sets should be replaced by an integral.

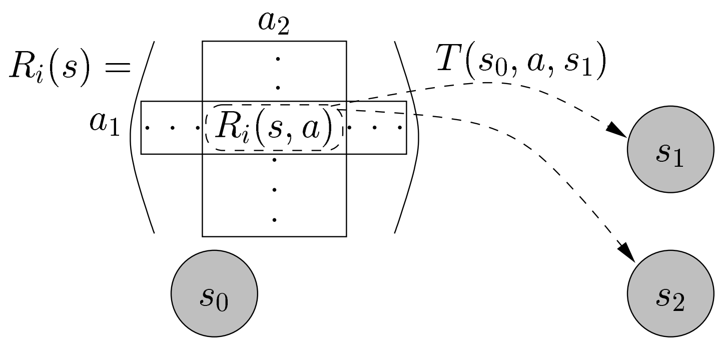


Figure 2.6: Graphical depiction of the stochastic game framework. Not all actions and transitions are shown.

2.4.1 Examples

Since MDPs and repeated games are both included in the framework of stochastic games, we have already given some examples of stochastic games. The single player grid soccer as well as all the matrix games in Table 2.1 are also examples of stochastic games. We introduce three specifically multiple state and multiple agent domains to further illustrate the model of stochastic games. These same games are also used in empirical results later in this work.

Grid Soccer

The first example is the full two player version of the example MDP described in Section 2.2.1. This version of grid soccer was first examined by Littman (1994). There are two players, marked **A** and **B**, that occupy non-identical locations from among the twenty grid cells. In addition, one player is in possession of the ball. This defines a state space, $\mathcal{S}$, where $|\mathcal{S}| = 20\times(20-1)\times 2 = 760$. Each player independently selects among the actions: **N**, **S**, **E**, **W**, and the hold action, **H**. As in the single player definition, these actions are then executed in random order. If an action causes the agent possessing the ball to enter its opponent's location, then the possession changes. Entering the other player's location otherwise has no effect. The player receives a reward of 1 for entering the goal with possession of the ball, and the other player receives a reward of -1. After a goal, the game is reset to one of the two initial states, where the players locations are shown in Figure 2.7 and possession is determined randomly. Notice that in the two-player definition, the next state is determined by the joint action of the players. An example of possible transitions and associated probabilities are shown in Figure 2.7. From any state and joint action, there are at most two possible next states, which are determined by the order of execution of the actions.

Grid Navigation

A non-competitive example of a stochastic game is Hu's grid navigation domain (Hu, 1999) shown in Figure 2.8. The agents start in two corners and are trying to reach the goal square on the opposite wall. They receive a reward of 1 for an action that enters this location, and a reward of 0 otherwise. The players have the four compass actions (i.e., **N**, **S**, **E**, and **W**), which are in most cases deterministic. If the two players attempt to move to the same square, both moves fail. To

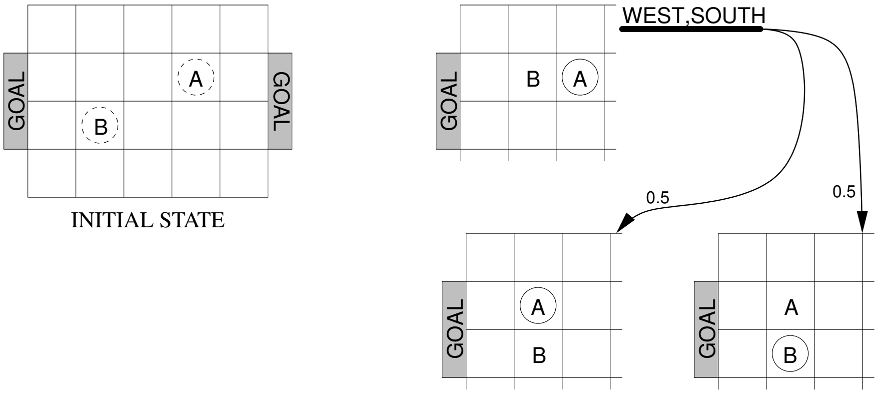


Figure 2.7: Two player grid soccer. The initial states are shown on the left, where possession is determined randomly. An example transition from a particular state and *joint* action is shown on the right. The outcome depends on the order that the actions are executed. If **A**'s action is executed first, it collides with **B** and loses the ball and then **B** moves south. Otherwise, **B** moves south first and **A** then moves west maintaining possession. Each outcome is equally likely.

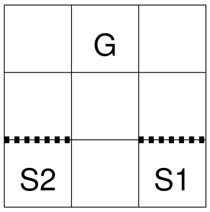


Figure 2.8: Grid navigation game. The dashed walls represent the transitions that are uncertain, i.e., trying to cross the walls will fail with probability $1/2$.

make the game interesting and force the players to interact, from the initial starting position the North action is uncertain, and it moves the player North with probability 0.5. Hence, the optimal path for each agent is to move laterally on the first move and then move North to the goal, but if both players move laterally then the actions fail. So the players must coordinate their actions.

Goofspiel

Goofspiel (or The Game of Pure Strategy) was invented by Merrill Flood while at Princeton (Flood, 1985). The game has numerous variations, but here we focus on the simple two-player, n-card version. Each player receives a suit of cards numbered 1 through n, a third suit of cards is shuffled and placed face down as the deck. For each round, the next card is flipped over from the deck, and the two players secretly select a card. They are revealed simultaneously and the player with the highest card wins the card from the deck, which is worth its number in points. If the players choose the same valued card, then neither player gets any points. Regardless of the winner of the round, both players discard their chosen card. This is repeated until the deck and players hands are exhausted. The winner is the player with the most points. The standard version of the game uses

n	VALUE(det)	VALUE(random)	$\lvert S\rvert$	$\lvert S \times A\rvert$	SIZEOF(π or Q)
4	-2	-2.5	692	15150	$\sim$ 59KB
8	-20	-10.5	3×10^6	1×10^7	$\sim$ 47MB
13	-65	-28	1×10^{11}	7×10^{11}	$\sim$ 2.5TB

Table 2.2: Facts about goofspiel. The table shows the number of states and state-actions, and the size of a stochastic policy or Q table for goofspiel depending on the number of cards, n. The VALUE columns list the worst-case value of the best deterministic policy and the random policy respectively.

all thirteen cards in a suit ($n = 13$), but smaller versions can also be considered.

This game can be described using the stochastic game model. The state is the current cards in the players' hands and deck along with the upturned card. The actions for a player are the cards in the player's hand. The transitions follow the rules as described, with an immediate reward going to the player who won the upturned card. Since the game has a finite end and we are interested in maximizing total reward, we use the discounted reward framework and set the discount factor γ to be 1. We discuss reward formulations in the next section.

This game has numerous interesting properties. First, notice that this game is zero-sum, and as with many zero-sum games any deterministic strategy can be soundly defeated. In this game, a deterministic strategy can be defeated by simply choosing the card valued one higher than the player's deterministically chosen card. The defeating strategy will win every card except one, and so the deterministic strategy will lose by at least sixty-five ($\sum_{i=1\ldots12} i - 13 = 65$). On the other hand, randomly selecting actions is a reasonably good policy in comparison. The defeating strategy for the random policy is to select the card matching the value of the upturned card (Ross, 1971). The random strategy, in this case, would have an expected loss of twenty-eight points. These values along with the same worst-case values for other sizes of the game are summarized in Table 2.2.

The third property is related to the tractability of solving the game. Notice that the number of states and state-action pairs grow exponentially with the number of cards. With the standard sized game, $n = 13$, the state space is so large that just storing one player's policy or Q-table would require approximately 2.5 terabytes of memory. Gathering data on all of the state-action transitions would require over 10^{10} playings of the game[4]. Table 2.2 shows the number of states and state-action pairs as well as the policy size for three different values of n.

This game epitomizes the need for approximation and generalization techniques as we discussed for the single-agent learning problem in Section 2.2.3. We return to this game in Chapter 6, where we present compelling results of learning in this challenging setting.

[4]Examining the number of states at each round, we find that it grows until the midpoint of the game is reached and then it contracts. Each game can only visit one state and action for each round, so the number of games must be at least the number of state-action pairs in the largest round. In the 13 card game, there are 3.5×10^{10} state-action pairs in round 7, thus requiring at least that many games to experience each transition. Of course, if the other player is not cooperating in the exhaustive search of the state space, many more games must be played.

2.4.2 Solution Concepts

Stochastic games borrow solution concepts from both MDPs and matrix games. Like MDPs, the goal for player i in a stochastic game is to learn a policy that maximizes long-term reward. A policy for player i, π_i is a mapping that defines the probability of selecting an action from a particular state. Formally, $\pi_i \in \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, where,

$$\forall s \in \mathcal{S} \qquad \sum_{a \in \mathcal{A}} \pi(s, a) = 1.$$

Since deterministic strategies can be exploited in matrix games, deterministic policies can also be exploited in stochastic games. Therefore, we cannot restricted ourselves to deterministic policies as is common with the study of MDPs. Throughout this work, we consider the full space of stochastic policies. We use π_i to refer to a policy for player i. We use π to refer to a joint policy for all the players, with π_i being player i's policy within that joint policy. We use the notation Π_i to be the set of all possible stochastic policies available to player i, and $\Pi = \Pi_1 \times \ldots \times \Pi_n$ to be the set of joint policies of all the players. We also use the notation π_{-i} to refer to a particular joint policy of all of the players except player i, and Π_{-i} to refer to the set of such joint policies. Finally, the notation $\langle \pi_i, \pi_{-i} \rangle$ refers to the joint policy where player i follows π_i while the other players follow their policy from π_{-i}.

Reward Formulations

The reward formulations of discounted reward and average reward also can be applied to stochastic games to quantify the value of a policy. In the discounted reward framework, the value of the joint policy π to player i at state, s, with discount factor, γ, is,

$$V_i^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t E\{r_i^t | s^0 = s, \pi\}, \tag{2.11}$$

where $E\{r_i^t | s^0 = s, \pi\}$ is the expected reward to player i received at time t given the initial state is s and the agents follow the joint policy π. Notice that this is nearly identical to Equation 2.1 for MDPs, except that each agent has a separate value function and it depends on the joint policy of the agents. Similarly, the average reward formulation in stochastic games is defined as,

$$V_i^{\pi}(s) = \lim_{T \to \infty} \sum_{t=0}^{T} \frac{1}{T} E\{r_i^t | s^0 = s, \pi\}. \tag{2.12}$$

As with MDPs, we can also define Q-values for a given player for a particular joint policy. For the discounted reward framework, this corresponds to,

$$Q_i^{\pi}(s, a) = R_i(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_i^{\pi}(s').$$

As with MDPs, it will often be much easier for learning algorithms to focus on Q-values instead of state values.

Best Response and Equilibria

Notice that a policy for a player can only be evaluated in the context of all the player's policies. This is the same difficulty faced by matrix games and the same solution concepts can be applied from matrix games. Since the set of stochastic games also contains all matrix games, we know that there is no optimal policy independent of the policies of the other players. The concept of best-response, though, can be defined.

Definition 4 *For a stochastic game, the* best-response function *for player i, $\mathrm{BR}_i(\pi_{-i})$, is the set of all policies that are optimal given the other player(s) play the joint policy π_{-i}. Formally, $\pi_i^* \in \mathrm{BR}_i(\pi_{-i})$ if and only if,*

$$\forall \pi_i \in \Pi_i \; \forall s \in \mathcal{S} \qquad V_i^{\left\langle \pi_i^*, \pi_{-i} \right\rangle}(s) \geq V_i^{\langle \pi_i, \pi_{-i} \rangle}(s).$$

We can also define the concept of a best-response equilibrium or Nash equilibrium.

Definition 5 *For a stochastic game, a* Nash equilibrium *is a collection of policies, one for each player, π_i, such that,*

$$\pi_i \in \mathrm{BR}_i(\pi_{-i}).$$

So, no player can do better by changing policies given that the other players continue to follow the equilibrium policy.

2.4.3 Types of Stochastic Games

The same classification for matrix games can be used with stochastic games. *Team games* are ones where all the agents have the same reward function. *Zero-sum games* are two-player games where one player's reward is always the negative of the other's. And *general-sum games* refer to all types of reward structures. In addition, stochastic games can also be classified based on their transition function. A *single-controller stochastic game* is a game where transition probabilities depend on the state and the action of only one of the players. Formally, this corresponds to,

$$\forall s, s' \in \mathcal{S} \; \forall a, a' \in \mathcal{A} \qquad a_1 = a'_1 \Rightarrow T(s, a, s') = T(s, a', s').$$

Similarly, in *no-control stochastic games*, no player's action affects the transition probabilities. Formally,

$$\forall s, s' \in \mathcal{S} \; \forall a, a' \in \mathcal{A} \qquad T(s, a, s') = T(s, a', s').$$

Like matrix games, zero-sum stochastic games have a unique Nash equilibrium, although finding this equilibrium is no longer as trivial. We examine techniques for solving stochastic games in Chapter 3.

2.4.4 The Markov Assumption

The Markov assumption from Markov decision processes still holds in stochastic games but it has a different form.

Definition 6 *A multiagent decision problem is* Markovian *if and only if, the sequence of states ($s^t \in \mathcal{S}$),* joint *actions ($a^t \in \mathcal{A}$), and rewards ($r_i^t \in \mathbb{R}$) satisfies*

$$Pr\left\{s^t = s, r_i^t = r_i | s^{t-1}, a^{t-1}, \ldots, s^0, a^0\right\} = Pr\left\{s^t = s, r_i^t = r_i | s^{t-1}, a^{t-1}\right\},$$

that is, if the next state and rewards depend only on the previous state and all *of the agents' actions, but not on the history of states and actions.*

From the game's perspective, stochastic games are Markovian, but from a single agent's perspective, the process is not Markovian. Specifically, the transition probabilities associated with a single agent's action from a state are not stationary and change over time as the other agents' action choices change. These choices are likely influenced by the past history of play, and so the history of play influences the future transition probabilities when revisiting a state. Therefore, from a single agent's perspective, the environment no longer appears stationary or Markovian.

This is one of the key challenges of learning in multiagent environments. The Markov assumption and the stationarity property are critical to single-agent reinforcement learning research. This violation of basic assumptions requires new techniques to be developed to learn effective policies in stochastic games.

2.4.5 Relationship to Matrix Games and MDPs

In addition to subsuming MDPs and matrix games, stochastic games have further connections to these simpler frameworks. Consider fixing stationary policies for all but one of the agents. The resulting decision process for the remaining agent is a Markov decision process. The MDP's states and the player's action set are the same as in the original stochastic game. The transition function is composed of the stochastic game's transition function with the other players' policies. Formally, let T be the stochastic game transition function and $\pi_{-i} \in \Pi_{-i}$ be a fixed policy for all the other agents, then the MDP transition function $\hat{T}$ is,

$$\hat{T}(s, a_i, s') = \sum_{a_{-i} \in \mathcal{A}_{-i}} \pi_{-i}(s, a_{-i}) T(s, \langle a_i, a_{-i} \rangle, s').$$

So, if the other agents are stationary and not learning, the problem simply reduces to the problem of learning to act in an MDP.

There is also a connection between matrix games and stochastic games. A Nash equilibrium policy in a stochastic game can be reduced to Nash equilibrium strategies in a set of associated state-specific matrix games. These matrix games are defined by the Q-values of the particular equilibrium policy. Formally, let π^* be a Nash equilibrium for a stochastic game. The joint strategy where player i plays action a_i with probability $\pi_i^*(s, a_i)$ is a Nash equilibrium for the matrix game where $R_i(a) = Q^{\pi^*}(s, a)$. This fact can be seen by observing that if π^* did not define a Nash equilibrium of the matrix game defined by $Q^{\pi^*}(s, \cdot)$, then some player could play a different action at state s to improve its strategy in that matrix game. This would also improve the value of the player's overall policy and so the joint policy is not an equilibrium of the overall stochastic game, proving the fact by contradiction. This relationship of stochastic game equilibria to equilibria in the game's state-wise matrices will be a useful fact for devising learning algorithms, as we will see in Chapter 3.

2.4.6 Behavioral Policies

In our presentation of repeated games and stochastic games we have restricted our attention to Markovian policies, i.e., policies whose action distributions depend only on the current state. This was intentional as this work will exclusively examine the learning of Markovian policies. However, these policies are a subset of a larger class of policies, called *behavioral policies*. Behavioral policies depend not only on the current state but also on the complete history of past states and joint actions.

Let $\mathcal{H}_k : (\mathcal{S} \times \mathcal{A})^k$ be the set of all possible histories of length k of past states and associated joint actions. Then, let, $\mathcal{H} = \cup_{i=0}^{\infty} \mathcal{H}_k$, be the union of possible finite length histories. A behavioral policy for player i, is a mapping, $\pi_i : \mathcal{H} \times \mathcal{S} \times \mathcal{A}_i \rightarrow [0, 1]$, where,

$$\forall h \in \mathcal{H} \quad \forall s \in \mathcal{S} \qquad \sum_{a \in \mathcal{A}} \pi(h, s, a) = 1.$$

Markovian policies are contained within this set as policies satisfying,

$$\forall h_1, h_2 \in \mathcal{H} \quad \forall s \in \mathcal{S} \qquad \pi(h_1, s, a) = \pi(h_2, s, a).$$

One could also consider the set of k-Markov policies, where the policy has the same distribution over actions for any histories that agree on the last k state and joint-action pairs. As with Markov policies, a notion of value for joint policies can also be defined for behavioral policies, using either a discounted or average reward formulation.

Repeated Games

Behavioral policies have been studied mostly in the framework of repeated games, where they are called behavioral strategies (Osborne & Rubinstein, 1994). Since repeated games have a single state, a behavioral strategy is a mapping from histories to the probability of selecting a particular action. It is a well-studied phenomenon in game theory that in repeated games there exist additional Nash equilibria when considering the space of behavioral strategies. These additional equilibria can also be strictly better than all Nash equilibria involving just Markovian strategies.

The canonical example of this is in the prisoner's dilemma. The game is shown in Table 2.3. Notice that the only Nash equilibrium of the game is the pure strategies where both players choose action **D**. In the repeated game with behavioral strategies, other equilibria exist. In particular, consider the *Tit-for-Tat* strategy, which is a 1-Markov behavioral strategy. The strategy plays **C** on the first playing, and afterward plays the same action its opponent played on the previous playing. For discounted reward with a suitably high value of γ or for average reward, Tit-for-Tat by both players is a Nash equilibrium. Also, it nets both players an average reward of 3, as opposed to the Markovian equilibrium where both receive a reward of 1.

Folk Theorems

There are a collection of theorems referred to as folk theorems, since their originator is unknown, that generalize the idea of Tit-for-Tat (Osborne & Rubinstein, 1994). The essence of these results is that under a variety of reward formulations, for any *strictly enforceable* and *feasible* set of payoffs

$$R_1 = \begin{array}{c c} & \begin{array}{cc} \mathbf{C} & \mathbf{D} \end{array} \\ \begin{array}{c} \mathbf{C} \\ \mathbf{D} \end{array} & \begin{pmatrix} 3 & 0 \\ 4 & 1 \end{pmatrix} \end{array} \qquad R_2 = \begin{array}{c c} & \begin{array}{cc} \mathbf{C} & \mathbf{D} \end{array} \\ \begin{array}{c} \mathbf{C} \\ \mathbf{D} \end{array} & \begin{pmatrix} 3 & 4 \\ 0 & 1 \end{pmatrix} \end{array}$$

Table 2.3: The prisoner's dilemma matrix game.

to the players, there exist equilibrium behavioral strategies that achieve these payoffs. Strictly enforceable means all players receive a payoff larger than their minimax optimal value (i.e., the value of the game if it were zero-sum). Feasible means the payoffs equal a convex combination of the payoffs associated with the joint actions of the game. The proofs involve constructing *trigger strategies* that play a particular sequence of joint actions (whose average or discounted value equals the target value), and punishing any deviance by the other player from this sequence by playing the strategy that forces them to get their minimax value. Since following along with the trigger gets a higher value, the other player has no incentive to deviate and be punished.

The power of the folk theorems is that certain communally beneficial play, such as both players choosing **C** in the prisoner's dilemma, can be justified as an equilibrium. The problem with the folk theorems is that any feasible and enforceable payoffs to the players can be justified as achievable by some behavioral strategy Nash equilibrium. In addition, from the perspective of prescribing a particular behavior, these equilibria really depend on an announcement of strategy. That is, the other player needs to know the specific trigger strategy being played in order to comply. In our learning paradigm there is no announcement and the game is unknown, and so requires exploration. It's not clear that these trigger strategies, or their subgame perfect variants (Osborne & Rubinstein, 1994), would operate well in these circumstances.

Our Focus on Markov Policies

As we have mentioned, this work will focus on Markovian policies. Although there are specific situations where behavioral policies can offer a strictly superior equilibrium solution, such as in prisoner's dilemma, they also add a number of complications. First, many games, such as zero-sum games, receive no benefit from the use of behavioral policies. Second, as the folk theorems demonstrate, there are an infinite number of equilibria in the space of behavioral strategies, with little justification for one over the other. Finally, behavioral polices severely increase the complexity of the learning problem. If one considers the number of states for an agent as distinguishable choice points, then the number of these grow exponentially as the k in k-Markov policies increases. For example, consider the grid navigation game in Section 2.4.1. The number of choice points for this game grows from 73 for 0-Markov policies, to 84,264 for 1-Markov polices, to approximately 10^8 for 2-Markov policies. In this domain, the history of very recent actions is probably not even particularly helpful. A more helpful class of behavioral policies would be ones that could make decisions contingent upon the last joint action selected in every state. Such a class of polices would need to define actions for well over 10^{100} states.

Obviously a line has to be drawn on the space of policies considered in order for learning to be tractable. Investigations in domains similar to prisoner's dilemma will want to examine a larger class of policies, such as k-Markov. Sandholm and Crites (1996) did just that when exploring reinforcement learning algorithms in iterated prisoner's dilemma. In the present work, we simply draw that line around Markovian policies. Although not a focus of this work, we believe many of the techniques described in this thesis still should have applicability in learning more general classes of policies.

2.5 Summary

In this chapter we presented the framework of stochastic games as a general model of multiagent interaction. We also described the subsumed frameworks of Markov decision processes, studied in the reinforcement learning community, and matrix games, studied in the field of game theory. We presented examples of problems using these models, including problems that will be the testbed for empirical experiments later in this work. We also introduced the key solution concepts and algorithms for MDPs: discounted and average reward, value functions, optimal policies, and simple reinforcement learning techniques. We presented an overview of the key solution concepts in matrix games: best-responses and Nash equilibria. Finally, we showed how stochastic games merges these two frameworks combining the challenges of learning to maximize reward over time with the strategic problems of multiple interacting agents. In the next chapter, we examine techniques for finding solutions and learning within the framework of stochastic games.

Chapter 3

Categorization of Previous Work

In this chapter we explore previous work on solving and learning in stochastic games. We also review work on agent limitations, focusing primarily on handling limitations within the context of decision making.

We begin by examining algorithms for *solving* stochastic games, which amounts to finding a Nash equilibrium for the game. These algorithms were developed within the field of game theory, and therefore do not concern themselves with mechanisms for learning to play an equilibrium. We present these algorithms because equilibria is an important concept for the learning paradigm, and they also have strong connections to individual learning algorithms.

We then explore the variety of algorithms for *learning* in stochastic games. We separate the algorithms into two classes: equilibrium learners and best-response learners. These algorithms differ on the specifics of what they are explicitly seeking to optimize. We make heavy use of this distinction in Chapter 4 when we introduce desirable properties of learning algorithms. We also discuss the algorithms' connections to their game theoretic counterparts for solving stochastic games.

Finally, we examine previous work in handling agent limitations in a decision making context.

3.1 Finding Equilibria

The first algorithm presented for finding a Nash equilibrium in a stochastic game was due to Shapley (1953) along with his statement of the stochastic game framework. The algorithm solved zero-sum games using an iterative dynamic programming process. Since then, numerous techniques have been proposed. Before we examine a few of these methods, we look at the basic assumptions and goals motivating these algorithms.

Game theory is concerned with Nash equilibria primarily as a predictive concept. That is, individually rational and knowledgeable agents will only play Nash equilibria. Finding equilibria is then important for the purpose of predicting behavior. The algorithms from game theory have some very strong assumptions.

1. The game, $(n, \mathcal{S}, \mathcal{A}_i, T, R_i)$, is fully known.

2. The goal is to compute the value of a Nash equilibrium, i.e., the expected discounted (or average) reward for all the players of playing an equilibrium.

1. Initialize V arbitrarily.
2. Repeat,
   (a) For each state, $s \in \mathcal{S}$, compute the matrix,
   $$G_s(V) = \left[g_{a \in \mathcal{A}} : \begin{array}{l} R(s,a)+ \\ \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V(s') \end{array} \right].$$
   (b) For each state, $s \in \mathcal{S}$, update V,
   $$V(s) \leftarrow \mathrm{Value}\left[G_s(V)\right].$$

Table 3.1: Shapley's value-iteration algorithm for finding equilibria in zero-sum stochastic games.

3. No concern is given to how this behavior might arise in actual multiagent interactions.

Learning algorithms, which we discuss later, reject most of these assumptions. Many of the learning algorithms, though, have a strong similarity to one of these "counterpart" methods from game theory. We note the similarities with these algorithms when we explore the specific learning algorithms. In addition, some of these algorithms also have a temporal differencing flavor, which we introduced in Chapter 2 in the context of solving MDPs. This connection provides further insight into the relationship between multiagent and single-agent techniques.

All of these algorithms use the concept of a value function over states, $V^\pi(s)$. Most of the techniques focus on zero-sum stochastic games, so there is a unique Nash equilibrium value function. The goal of these algorithms is to converge to this "optimal" value function, $V^*(s)$.

We first explore Shapley's original iterative algorithm for zero-sum games, and then examine two other iterative techniques: Pollatschek and Avi-Itzhak, and Van der Wal. We then examine how the game can be solved as a linear program. Finally, we look at fictitious play, that involves "playing" the stochastic game to find a solution.

3.1.1 Shapley

Shapley's algorithm (1953) for computing a Nash equilibrium in a zero-sum stochastic game was a direct result of his existence proof of equilibria in these games. The algorithm is shown in Table 3.1. The algorithm uses a temporal differencing technique to backup values of next states into a simple matrix game, $G_s(V)$. The value function V is then updated by solving the matrix game for a Nash equilibrium value at each state. Since the stochastic game is zero-sum, these intermediate matrix games are also zero-sum, and therefore have a unique value.

Notice the algorithm is nearly identical to value iteration for MDPs, with the "$\max$" operator replaced by the "Value" operator. The algorithm also shows that equilibria in stochastic games are solutions to Bellman-like equations. The algorithm's value function V converges to V^* which satisfies the following equations,

$$\forall s \in \mathcal{S} \qquad V^*(s) \;=\; \mathrm{Value}\left[G_s(V^*)\right]. \tag{3.1}$$

1. Initialize V arbitrarily.
2. Repeat,

$$\begin{aligned} \pi_i &\leftarrow \forall s \in \mathcal{S} \quad \text{Solve}_i\ [G_s(V)] \\ V(s) &\leftarrow E\left\{\sum \gamma^t r_t | s_0 = s, \pi_i\right\}. \end{aligned}$$

Table 3.2: Pollatschek & Avi-Itzhak's policy-iteration algorithm for finding equilibria in zero-sum stochastic games. The function G_s is the same as presented in Table 3.1.

The crucial fact in proving convergence of V is to show that the value function mapping in Step 2 of Table 3.1 is a contraction mapping under the max-norm. Let PV be the value function resulting from applying Step 2 to the function V. Formally, P is a contraction mapping under the max norm if and only if,

$$||PV - PV^*||_{\max} < \gamma ||V - V^*||_{\max}. \tag{3.2}$$

Since V^* satisfies Equation 3.1 then $PV^* = V^*$. It is fairly easy to see that this condition is enough to show that repeated applications of Step 2 results in V's decreasing distance to V^*, resulting in convergence.

3.1.2 Pollatschek & Avi-Itzhak

Just as Shapley's algorithm is an extension of value iteration to stochastic games, Pollatschek & Avi-Itzhak (Pollatschek & Avi-Itzhak, 1969) introduced an extension of policy iteration (Howard, 1960). The algorithm is shown in Table 3.2. Each player selects the equilibrium policy according to the current value function, making use of the same temporal differencing matrix, $G_s(V)$, as in Shapley's algorithm. The value function is then updated based on the actual rewards of following these policies.

Like Shapley's algorithm, this algorithm also computes the equilibrium value function, from which can be derived the equilibrium policies. This algorithm, though, is only guaranteed to converge if the transition function, T, and discount factor, γ, satisfies a certain property. In particular, $\gamma < 1/3$, or the following holds,

$$\max_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} \left(\max_{a \in \mathcal{A}} T(s, a, s') - \min_{a \in \mathcal{A}} T(s, a, s') \right) < \frac{1 - \gamma}{\gamma}.$$

This amounts to a limit on the overall effect the players' actions can have on the transition function. For example, the extreme of no-control stochastic games satisfies this property.

A similar algorithm by Hoffman & Karp (Hoffman & Karp, 1966) which alternates the policy learning, rather than it occurring simultaneously, avoids the added assumption. Obviously, this requires control over when the other agents in the environment are learning, and so limits the usefulness of the technique from our learning perspective.

1. Initialize V arbitrarily.
2. Repeat,
   (a) Compute an optimal policy for player two,
   $$\pi_2 \leftarrow \forall s \in \mathcal{S} \quad \mathrm{Solve}_i\left[G_s(V)\right].$$
   (b) Repeat λ times, $\forall s \in \mathcal{S}$,
   $$V(s) \leftarrow \max_{a_1 \in \mathcal{A}_1} \sum_{a_2 \in \mathcal{A}_2} \pi_2(s, a_2) \left(R(s, \langle a_1, a_2 \rangle) + \gamma \sum_{s' \in \mathcal{S}} T(s, \langle a_1, a_2 \rangle, s') V(s') \right).$$

Table 3.3: Van der Wal's class of algorithms depending on the value of the parameter, $\lambda \in \mathbb{N} \cup \{\infty\}$. If $\lambda = 1$ then the algorithm is equivalent to Shapley's algorithm in Table 3.1. If $\lambda = \infty$ then the algorithm is the Hoffman and Karp variant of Pollatschek & Avi-Itzhak's algorithm in Table 3.2.

3.1.3 Van der Wal

The similarity of Shapley's algorithm to value iteration, and the Pollatschek and Avi-Itzhak algorithm to policy iteration, suggests a mixture of the two techniques might be possible. This is precisely the family of algorithms explored by Van der Wal (Van der Wal, 1977) using a integer parameter $\lambda \in \mathbb{N} \cup \{\infty\}$. This parameter determines the number of steps of lookahead in computing the new value function. The λ serves a similar purpose to the parameterized temporal differencing algorithms, such as Sarsa($\lambda \in [0, 1]$), providing a continuum between a pure policy iteration approach, and a pure value iteration approach.

Like Shapley and Hoffman and Karp, V converges to V^*, and equilibrium policies can be constructed for both players by solving $G_s(V^*)$.

3.1.4 Linear and Nonlinear Programming

As we discussed in Section 2.3.3, equilibria can be computed for zero-sum matrix games by solving a particularly constructed linear program. For two-player general-sum matrix games, an equilibrium can be found as a solution to a nonlinear program with a quadratic objective function and linear constraints (Mangasarian & Stone, 1964).

General mathematical programming techniques can also be extended to stochastic games. For zero-sum stochastic games, the objective functions and constraints remain linear. For two-player general-sum stochastic games, the solution now requires solving a nonlinear complementarity problem. Filar and Vrieze (Filar & Vrieze, 1997) give a thorough treatment of mathematical programming in stochastic games, both for zero-sum and general-sum. Although, these are highly efficient methods for finding equilibria in known games, they do not fit well with the learning assumptions of incremental experience.

3.1.5 Fictitious Play

The final algorithm for finding equilibria in a known stochastic game is the technique of fictitious play. This was first proposed as a technique for finding equilibria in matrix games (Brown, 1949; Robinson, 1951) and was later extended to stochastic games (Vrieze, 1987). Fictitious play is also the algorithm that most easily fits into a learning paradigm. The stochastic game version of the algorithm is shown in Table 3.4.

Fictitious play assumes opponents play stationary strategies. The algorithm maintains information about the average estimated sum of future discounted rewards. This is computed by maintaining a total of the estimated future discounted reward that would have been received if that action was always selected from that state, $Q_i(s, a)$[1]. The algorithm then deterministically selects the action for each player that would have done the best in the past, according to $Q_i(s, a)$. The estimated sum of future discounted rewards is computed using a simple temporal differencing backup.

1. Initialize $Q_i(s \in \mathcal{S}, a_i \in \mathcal{A}_i)$ according to specific constraints, and $t \to 0$.
2. Repeat for each state, $s \in \mathcal{S}$,
   (a) Let $a_i = \operatorname{argmax}_{a_i \in \mathcal{A}_i} Q_i(s, a_i)$.
   (b) Update $Q_i(s, a_i), \forall s \in \mathcal{S}\, \forall a'_i \in \mathcal{A}_i$

   $$Q_i(s, a'_i) \leftarrow Q_i(s, a'_i) + R(s, \langle a_{-i}, a'_i \rangle) + \gamma \left(\sum_{s' \in \mathcal{S}} T(s, a, s') V(s') \right)$$

   where,

   $$V(s) = \max_{a_i \in \mathcal{A}_i} \frac{Q_i(s, a_i)}{t}.$$

   (c) $t \to t + 1$.

Table 3.4: Fictitious play for two-player, zero-sum stochastic games using a model.

The algorithm has a number of nice properties. It is capable of finding equilibria in both zero-sum games and some classes of general-sum games, in particular iterated dominance solvable games (Fudenberg & Levine, 1999). These are games where a repeated process of eliminating strictly dominated actions leaves only a single policy for both players, which must be a Nash equilibrium (Osborne & Rubinstein, 1994).

The algorithm is also very amenable to a more explicit agent setting. We can easily separate the algorithm into separate processes, one for each agent, where each agent selects its action that maximizes its own $Q_i(s, a_i)$, observes the action selected by the other agent and updates $Q_i(s, a_i)$. If both players are following fictitious play then this is equivalent to the described algorithm.

[1]Although not strictly a Q-value in the sense of Q-learning, it still serves to measure the relative value of selecting various actions from various states.

Matrix Game Solver	+	Temporal Differencing	=	Stochastic Game Solver

MG	+	TD	=	Solving
LP		TD(0)		Shapley
LP		TD(1)		Pollatschek & Avi-Itzhak
				Hoffman & Karp
LP		TD(λ)		Van der Wal
FP		TD(0)		Fictitious Play

LP: linear programming FP: fictitious play
QP: quadratic programming

Table 3.5: Summary of algorithms for solving stochastic games. Each contains a matrix game solving component and a temporal differencing component. "Solving" algorithms assume the transition and reward function are known.

Its drawback is that for zero-sum games, it can only *find* an equilibrium policy, without actually *playing* according to that policy. One obvious reason for this drawback is that the algorithm is deterministic, and cannot play stochastic policies. Since many zero-sum games do not have pure policy equilibria, it cannot play according to any equilibrium policy. There is an approximation to fictitious play for matrix games called smooth fictitious play (Fudenberg & Levine, 1999), which is capable of playing mixed equilibrium. This variation, though, has yet to be generalized to stochastic games.

3.1.6 Summary

With the exception of the mathematical programming methods, these algorithms have a similar structure. They all have a matrix game solving component combined with a temporal differencing component. This decomposition is summarized in Table 3.5. We now examine algorithms that learn a policy through observations of the game's transitions and rewards. Some of direct extensions of the equilibrium finding algorithms to conform to the basic learning assumptions.

3.2 Learning Algorithms

The focus of algorithms from game theory is to compute the value to the players, and possibly associated policies, of a Nash equilibrium for the stochastic game. Reinforcement learning presents a different paradigm where the goal is to *learn* through interaction rather than *solve* for an equilibrium. In this paradigm, it is generally assumed that the game being played, specifically T and R_i are not known but must be observed through experience. Agents are required to select actions in the environment in order to gain observations of T and R_i. The second distinguishing characteristic is that these algorithms focus on the behavior of a single agent, and seek to learn a policy specifically for that agent. The play for other agents, especially in zero-sum games, is not under the agent's control. There is still a great degree of variation in whether the algorithms are required to observe the actions selected by the other agents or the other agent's rewards.

We distinguish between to two broad classes of learning algorithms, with very different explicit goals. Equilibrium learners explicitly seek to estimate and converge to their policy in one of the game's Nash equilibria. Best-response learners seek to directly learn and play a best-response to the other players' policies. Although, not explicitly related to equilibria, best-response learners have a strong connection to equilibria, which we explore further in Chapter 4 when we consider *rationality* as a property of learning algorithms. These two classes of algorithms are then followed by a brief discussion.

3.2.1 Equilibrium Learners

Equilibrium learners seek to learn and play a Nash equilibrium of the stochastic game. Since an equilibrium is actually a joint policy, they actually seek to play their component of some Nash equilibrium. For zero-sum games this insures optimal worst-case performance, and also guarantees convergence to a Nash equilibrium against other equilibrium-playing opponents. For general-sum games there is the difficulty of multiple equilibria creating an equilibrium selection problem. This problem is ignored by most algorithms, assuming that there is either a unique equilibrium or there is some other mechanism for solving the equilibrium selection problem. As with zero-sum games, playing against other equilibrium-playing agents insures that the policies convergence to the game's unique (or selected) equilibrium.

There has been a line of research over the past decade in regards to the development of equilibrium learning algorithms, as well as determining their conditions for convergence. We review the continuing development of these techniques. It is useful, though, to note that all of the algorithms have a nearly identical structure. This is shown in Table 3.6. The algorithms differ mainly on the final line of Step 2(b) by using different definitions of the Value operator.

Minimax-Q

Littman (1994) extended the traditional Q-Learning algorithm for MDPs to zero-sum stochastic games. The notion of a Q function is extended to maintain the value of *joint* actions, and the backup operation computes the value of states differently. Instead of using the "$\max_{a\in AA_i}$" operator in Step 2(b) of Table 3.6, it computes the unique equilibrium value of the matrix game defined by the Q values at the current state. This is the meaning of the "Value" operator in the table. The Value operator is efficiently computed using the linear program from Section 2.3.3 for solving zero-sum matrix games. It is interesting to note that this is basically the off-policy reinforcement learning equivalent of Shapley's "value iteration" algorithm.

This algorithm does, in fact, converge to the stochastic game's equilibrium solution, assuming the other agent executes all of its actions infinitely often (Littman & Szepesvári, 1996). This is true even if the other agent does not converge to the equilibrium, and so provides an *opponent-independent* method for learning an equilibrium solution.

1. Initialize $Q(s \in \mathcal{S}, a \in \mathcal{A})$ arbitrarily, and set α to be the learning rate.

2. Repeat,

   (a) From state s select action a_i that solves the matrix game u$[\, Q(s,a)_{a \in \mathcal{A}}\,]$, with some exploration.

   (b) Observing joint-action a, reward r, and next state s',

   $$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma V(s')),$$

   where,

   $$V(s) = \text{Value}\left(\left[\, Q(s,a)_{a \in \mathcal{A}}\,\right]\right).$$

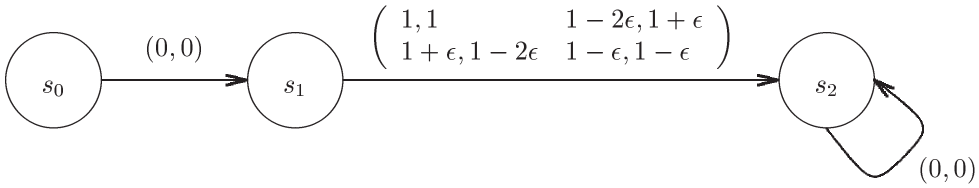
Table 3.6: Equilibria Learning algorithm. The differences between Minimax-Q, Nash-Q, FFQ, and CE-Q are in the Value function and the Q values. Minimax-Q uses the linear programming solution for zero-sum games, Nash-Q uses the quadratic programming solution for general-sum games, FFQ and CE-Q use specialized linear programming solutions. Also, the Q values in Nash-Q and CE-Q are actually a vector of expected rewards, one entry for each player.

Nash-Q

Hu & Wellman (1998) extended the Minimax-Q algorithm to two-player general-sum stochastic games. The algorithm is structurally identical and is also shown in Table 3.6. The extension requires that each agent maintain Q values for all of the agents. Also, the linear programming solution used to find the equilibrium of zero-sum games is replaced with the quadratic programming solution for finding an equilibrium in two-player general-sum games.

This algorithm is the first to address the complex problem of general-sum stochastic games. But the algorithm requires a number of very limiting assumptions. The most restrictive of which limits the structure of *all* the intermediate matrix games faced while learning, i.e., defined by the values $Q(s, a \in \mathcal{A})$. Specifically, all of these intermediate games must have a single equilibrium. In addition, the equilibrium in all of these intermediate games must be a global optimum, or the equilibrium in all of these intermediate games must be a saddle point. A global optimum is a joint action that maximizes each agent's payoff. A saddle point equilibrium is essentially an adversarial equilibrium. If a player deviates, it not only gets a lower payoff (the equilibrium criterion), but the other player necessarily gets a higher, or equal, payoff. This is a non-trivial restriction since it is not possible to predict whether the assumption remains satisfied while learning.

Convergence Difficulties. The general convergence difficulties of Nash-Q can actually be attributed to the fact that value iteration with general-sum games is not a contraction mapping under the max-norm (Bowling, 2000). Remember from Section 3.1.1 that P is a contraction mapping if the value function, V, is guaranteed on each application of P to move closer to V^* by a factor of γ. We consider the backup, P, for which Step 2(b) of Table 3.6 is a stochastic approximation.



$$Q^*(s_0) = (\gamma(1-\epsilon), \gamma(1-\epsilon)) \qquad Q^*(s_1) = \begin{pmatrix} 1,1 & 1-2\epsilon, 1+\epsilon \\ 1+\epsilon, 1-2\epsilon & \boxed{1-\epsilon, 1-\epsilon} \end{pmatrix} \qquad Q^*(s_2) = (0,0)$$

Figure 3.1: Counterexample to Nash-Q being, in general, a contraction mapping. The diagram shows a three state stochastic game, including the Nash equilibrium values, Q^*. All transitions are deterministic and are independent of the actions selected. The only choice available to the agents is in state s_1, where the corresponding matrix game has a unique Nash equilibrium where both players choose their second action.

Specifically, for two players this is,

$$PQ(s,a) = (R_1(s,a), R_2(s,a)) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') \, \text{Value}\left(\left[Q(s',a')_{a' \in \mathcal{A}} \right] \right).$$

Consider the stochastic game with three states shown in Figure 3.1. State s_0 always transitions to state s_1 with rewards 0. State s_1 is a 2×2 game with all actions causing the game to transition to a state, s_2, where no further rewards can be received. The rewards are shown above each transition, with commas separating the rewards to each player.

Remember that Nash-Q maintains a Q-value for each player. This is represented in the Q-values by a comma separating the values to player 1 and player 2. Consider the following Q function,

$$\begin{aligned} Q(s_0, \cdot) &= (\gamma, \gamma) \\ Q(s_1, \cdot) &= \begin{pmatrix} \boxed{1+\epsilon, 1+\epsilon} & 1-\epsilon, 1 \\ 1, 1-\epsilon & 1-2\epsilon, 1-2\epsilon \end{pmatrix} \\ Q(s_2, \cdot) &= (0,0). \end{aligned}$$

The matrix game corresponding to $Q(s_1)$ has a unique Nash equilibrium where both players choose their first action. Notice that $||Q - Q^*||_{\max} = \epsilon$. If P is a contraction mapping then after applying P the values should be closer to Q^*. The actual values for PQ are,

$$\begin{aligned} PQ(s_0, \cdot) &= (\gamma(1+\epsilon), \gamma(1+\epsilon)) \\ PQ(s_1, \cdot) &= \begin{pmatrix} 1,1 & 1-2\epsilon, 1+\epsilon \\ 1+\epsilon, 1-2\epsilon & \boxed{1-\epsilon, 1-\epsilon} \end{pmatrix} \\ PQ(s_2, \cdot) &= (0,0). \end{aligned}$$

Notice that $||PQ - PQ^*||_{\max} = 2\gamma\epsilon$, which is greater than ϵ if $\gamma > 0.5$. Hence with the max norm, P is not a contraction mapping.

This fact demonstrates that it is not just the non-uniqueness of equilibria in general-sum games that makes convergence difficult. In the above example, all of the matrix games have a single equilibrium, but the Nash-Q mapping is still not a contraction mapping. This is the reason Nash-Q's additional restrictions on the structure of all intermediate matrix games is necessary, and also brings the general applicability of Nash-Q into question.

Friend-or-Foe-Q

Littman's Friend-or-Foe-Q (FFQ) (2001) is an equilibrium learner that extends Minimax-Q to include a small class of general-sum games. Motivated by the assumptions of Nash-Q, which required that either the game be effectively zero-sum, so each intermediate game has a saddle point equilibrium, or the game was a team game, so each intermediate game has a global optimum. This extension handles both of these classes of games, as well as others, that do not by themselves fit under the Nash-Q assumptions.

The algorithm, in essence, assumes there are two competing teams of agents in the stochastic game. So, from one agent's perspective, every other agent is either a friend or a foe. Knowing the labeling, or inferring the labeling from observed rewards, the agent can alter the Minimax-Q value computation to account for these labels.

The Value computation uses the basic zero-sum linear program, but performs the program over the joint actions of all the agents that are on the same team. The result is a joint action for one team that maximizes their minimum reward received over all joint actions of the other team. The action for the agent is then their component in this equilibrium joint action. Like Minimax-Q, FFQ is guaranteed to converge to their policy in an equilibrium for the stochastic game. Although, since cooperative games may have multiple equilibria where policies are not interchangeable, in practice this algorithm can suffer from an equilibrium selection problem. With an agreed upon team convention or arbiter for the equilibrium selection problem, then the team can be guaranteed to be playing according to the same joint policy equilibrium.

CE-Q

The final equilibrium learning technique is Greenwald & Hall's Correlated-Q (CE-Q) (2002) that seeks to learn to play according to an equilibrium by using the broader class of *correlated equilibria*. Nash equilibria are independent stochastic distributions over player's actions. Correlated equilibria allow for stochastic distributions over joint actions, where players do no randomize independently. A correlated joint policy is a mapping from states and joint actions to probabilities. Formally, $\pi \in (\mathcal{S} \times \mathcal{A}) \rightarrow [0, 1]$, where,

$$\forall s \in \mathcal{S} \qquad \sum_{a \in \mathcal{A}} \pi(s, a) = 1.$$

To play according to a correlated policy, players must have access to a shared randomizer. This may be a central arbitrator that informs each player of the action they are to execute in order to follow the correlated policy.

A correlated joint policy is an equilibrium if and only if, for each player, i, each state, s, and each action, a_i, the following holds. Let $\sigma_{-i}(a_{-i}|a_i)$ be the conditional probability that the other agents select joint action a_{-i}, given agents are following the correlated joint policy π and agent i is playing action a_i. Then, for all a'_i, the following must be true,

$$\sum_{a_{-i}\in\mathcal{A}_{-i}} \sigma_{-i}(a_{-i}|a_i)Q^{\pi}(s,\langle a_i,a_{-i}\rangle) \geq \sum_{a_{-i}\in\mathcal{A}_{-i}} \sigma_{-i}(a_{-i}|a_i)Q^{\pi}(s,\langle a'_i,a_{-i}\rangle).$$

In other words, given the knowledge about the other player's distribution gained from one agent's own prescribed action, that agent gains no increase in expected payoff by playing an action different from its prescribed action.

All Nash equilibria are correlated equilibria, so this is a more general solution concept. In addition, for matrix games, correlated equilibria are far easier to find and involve solving a linear program, even in the case of general-sum games.

CE-Q is the straightforward replacement of the Value function in Table 3.6 with a method to compute the value of some correlated equilibrium. Like the Nash equilibrium, though, this concept is not unique and so the agents must use some central arbitrator to insure they all select the same equilibrium when computing the value of a state. This technique is more efficient than Nash-Q, since it does not require the complex quadratic programming Nash equilibrium solver. On the other hand, it requires some agreed upon equilibrium selection and shared randomizer in learning and execution, respectively. Greenwald & Hall do point out that there are recent algorithms for independent agent's empirical play to converge to a correlated equilibria in a repeated game. These online techniques may also be able to remove the need for a central arbitrator to act as a shared randomizer.

Learning in Team Games

Although, not the focus of this work, there has also been recent work on the problem of learning to play in a team stochastic game, where all agents have an identical reward function. The challenge of learning to play a Nash equilibrium in these games is not difficult in practice. Even single-agent learning techniques can achieve this goal (Claus & Boutilier, 1998). However, the team game assumption opens up the possibility of stronger guarantees. In particular, Wang and Sandholm's (2002) Optimal Adaptive Learning is guaranteed to converge in self-play to the *optimal* Nash equilibrium in any team stochastic game. That is, the play converges to strategies that achieve the highest possible payoffs, thus avoiding all suboptimal Nash equilibria, which other algorithms often converge to. Brafman and Tennenholtz (2002a) go further, presenting an algorithm that with arbitrarily high probability will converge in self-play to an optimal Nash equilibrium in *polynomial time*. In summary, the assumption that the game is a team game is very powerful for making stronger guarantees on the learned policies.

3.2.2 Best-Response Learners

The second class of algorithms we examine do not explicitly consider equilibria. Instead, they simply attempt to learn a best-response to the other player's current policies. A major consideration for looking at best-response learning is that agents are not always fully rational. Playing an equilibrium policy is only sensible when the other agents also play according to the equilibrium. When considering limited agents, they may not be capable of learning or playing the equilibrium policy. Best-response algorithms have the possibility of both coping with limited teammates as well as exploiting limited opponents.

The first best-response learning algorithm we examine is the ubiquitous single-agent learning algorithm, Q-learning. We then explore a learning variation of fictitious play, gradient techniques, and the class of learners known as *regret-minimizing* algorithms.

Q-Learning

Q-Learning (Watkins, 1989) is a single-agent learning algorithm specifically designed to find optimal policies in MDPs. In spite of its original intent it has been widely used for multiagent learning, and not without success (Tan, 1993; Sen, Sekaran, & Hale, 1994; Sandholm & Crites, 1996; Claus & Boutilier, 1998). It also has some theoretical merits. As observed, when the other players play a stationary strategy, the stochastic game "becomes" an MDP, and therefore Q-learning learns to play an optimal response to the other players.

However, Q-learning traditionally cannot learn or play stochastic policies. This prevents Q-learners from converging to equilibria solutions for the class of zero-sum games, since they may only have mixed policy equilibria, such as matching pennies and rock-paper-scissors from Table 2.1. There are single-agent learning techniques, similar to Q-learning, that are capable of playing stochastic policies (Jaakkola, Singh, & Jordan, 1994; Baird & Moore, 1999; Sutton et al., 2000; Baxter & Bartlett, 2000). These techniques mainly address the issues of partial observability or function approximation in single-agent decision problems. There has been no investigation of whether such techniques might perform well in multiagent environments.

An algorithm similar to Q-learning has also been investigated within the game theory community to explore convergence to equilibria (Jehiel & Samet, 2001). They examined a class of stochastic finite games where the transitions for any given state are deterministic and controlled by a single player. In addition, states are never revisited and rewards are only given for the terminating state, which is reached in a finite number of steps. They demonstrated that a simple strategy of choosing the action that maximizes the current value estimate, along with a simple Monte Carlo value update, will converge to optimal play. Here, optimal play means a strategy that guarantees a win, if such a guarantee is possible. The algorithm essentially amounts to Q-learning with $\alpha = 1$, while using episode replay to propagate values quickly through the Q-table. This further demonstrates that Q-learning can often be quite successful, particularly in games that do not require stochastic play.

Opponent Modelling

Opponent modelling Q-learning (Uther & Veloso, 1997) or joint action learners (JALs) (Claus & Boutilier, 1998) are reinforcement learning algorithms that are similar to fictitious play. The algorithm is shown in Table 3.7. Explicit models of the opponents are learned as stationary distributions over their actions (i.e., $C(s, a_{-i})/n(s)$ is the probability the other players will select joint action a_{-i} based on past experience). These distributions, combined with learned joint-action values from standard temporal differencing, are used to select an action. Uther & Veloso investigated this algorithm in the context of a zero-sum domain, and Claus & Boutilier examined it in the setting of team matrix games.

1. Initialize Q arbitrarily, and $\forall s \in \mathcal{S} \quad C(s) \leftarrow 0$ and $n(s) \leftarrow 0$.
2. Repeat,
   (a) From state s select action a_i that maximizes,
   $$\sum_{a_{-i}} \frac{C(s, a_{-i})}{n(s)} Q(s, \langle a_i, a_{-i} \rangle)$$
   (b) Observing other agents' actions a_{-i}, reward r, and next state s',
   $$\begin{aligned} Q(s,a) &\leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma V(s')) \\ C(s, a_{-i}) &\leftarrow C(s, a_{-i}) + 1 \\ n(s) &\leftarrow n(s) + 1 \end{aligned}$$
   where,
   $$\begin{aligned} a &= (a_i, a_{-i}) \\ V(s) &= max_{a_i} \sum_{a_{-i}} \frac{C(s, a_{-i})}{n(s)} Q(s, \langle a_i, a_{-i} \rangle). \end{aligned}$$

Table 3.7: Opponent modelling Q-learning algorithm

The algorithm has very similar behavior to fictitious play. It requires observations of the opponents' actions, but not of their individual rewards. Like fictitious play, its empirical distribution of play may converge to an equilibrium solution, but its action selection is deterministic and does not play mixed strategies.

Infinitesimal Gradient Ascent

The final best-response learning algorithm we examine is Infinitesimal Gradient Ascent (IGA) (Singh, Kearns, & Mansour, 2000b). Since this technique is highly related to the approach taken throughout this work, the algorithm is described in detail in Chapter 4. We give a brief summary here for completeness. The basic idea is for an agent to adjust its policy in the direction of the gradient of the value function. Against a stationary opponent, the policy will over time converge to a local maximum. Since the value function is linear in the space of policy parameters, all local maxima are global maxima, and so it necessarily converges to a best-response.

The algorithm was originally defined for only a two-player, two-action matrix games, where the reward function, R_i, is known. The concept, though, has also been applied as a single-agent learning technique to multiple state MDPs and POMDPs. These multiple state algorithms can scale to very large state spaces by learning within a parameterized subset of the complete policy space (Sutton et al., 2000; Baxter & Bartlett, 2000), and performing gradient ascent on the policy's parameters. The original analysis of IGA demonstrated that, not only does it learn best-responses, the average reward of two agents in self-play over time converges to the value of some Nash equilibrium for the game. These results are stated more formally in Chapter 4.

Regret-Minimizing Algorithms

There is another category of algorithms that do not directly learn an equilibrium nor learn and play best-responses, but rather seek to minimize *regret*. The main property of these algorithms is called *no-regret*, also known as *universal consistency* or *Hanan consistency*. These algorithms have primarily been investigated in single-state games, such as k-armed bandit problems and matrix games, and so we restrict our discussion to the single-state frameworks.

Regret at time T is the difference between the total reward received in T playings and the value of the best stationary strategy over those same playings. Consider the context of a repeated matrix game for player i. Let r_t be the actual value the player received at time, and let $N_T(a_{-i})$ be the number of times the other players played joint action a_{-i} in the first T trials. Then, mathematically, regret at time T is,

$$\text{Regret}_i(T) = \max_{a_i \in \mathcal{A}_i} \left(\sum_{a_{-i} \in \mathcal{A}_{-i}} N_T(a_{-i}) R_i(\langle a_i, a_{-i} \rangle) \right) - \left(\sum_{t=1}^{T} r_t \right).$$

In other words, regret is the difference between the total reward the agent received and the amount of reward the agent could have received had the player known the distribution of actions but not the order.

An algorithm achieves no-regret if and only if,

$$\lim_{T \to \infty} \text{Regret}_i(T) \leq 0.$$

So, the algorithm must do at least as well as if the player had known the ahead of time the distribution of the other agents' actions. Notice that against a fixed strategy, the algorithm is guaranteed to converge in payoffs to the value of the best-response strategy. Therefore, we have classified this algorithm with other best-response learning methods. Notice, though, that the value of the best action for a given distribution is at least as high as the minimax-optimal value of the game, if the game were zero-sum. Therefore, in zero-sum games, no-regret algorithms guarantee at least the Nash equilibrium value of the game. If the opponent, though, plays suboptimally, the actual total reward received may be higher.

Techniques that achieve this property have been discovered, and rediscovered numerous times (Fudenberg & Levine, 1999). One of the most well-known no-regret algorithm is Littlestone and Warmuth's randomized weighted majority (1994). A more recent algorithm by Auer and colleagues (1995) relaxed the requirement that payoffs for unplayed actions were observable. More recently, Hart and Mas-Collel (2002) introduced an algorithm with an even stronger no-regret property. They compare the total reward received with the amount of reward the agent could have received if instead of playing some action, a_i, it played a different action, a'_i, for the particular history of play. This is a strictly stronger notion of regret, which they demonstrate can also be achieved.

There has been comparatively little work applying these concepts to models that include state, such as MDPs or stochastic games. One exception for stochastic games is the work of Mannor (2001) of extending no-regret properties to average-reward stochastic games. There has been no work to generalize these approaches to discounted reward stochastic games.

3.2.3 Summary

Like the algorithms from the game theory literature, many of the learning techniques have an easily distinguishable matrix game solving component and temporal differencing component. This is summarized in the "Learning" column of Table 3.5.

One of the observations that can be made from Table 3.5 is the lack of an RL algorithm with a multiple step backup (i.e., policy iteration or TD(λ)). One complication for multi-step backup techniques is that they are *on-policy*, that is they require that the agent be executing the policy it is learning. In a multiagent environment, this would require that the other agents also be learning *on-policy*, which may not be the case. Despite this limitation it is still very suggestive of new algorithms that may be worth investigating. Also, there has been work to make an off-policy TD(λ) (Peng & Williams, 1996; Watkins, 1989), which may be useful in overcoming this limitation.

Finally, the distinction between equilibrium learners and best-response learners is very useful. In Chapter 4 we present two important properties of multiagent learning algorithms. Generally speaking, each class of algorithms achieves one property but not the other. The properties also demonstrate an additional connection between the two algorithm classes by demonstrating the relationship of best-response learners to the concept of equilibria.

Matrix Game Solver	+	Temporal Differencing	=	Stochastic Game Solver

MG	+	TD	=	Solving	Learning
LP		TD(0)		Shapley	Minimax-Q Friend-or-Foe-Q
LP		TD(1)		Pollatschek & Avi-Itzhak Hoffman & Karp	–
LP		TD(λ)		Van der Wal	–
QP		TD(0)		–	Nash-Q
CE		TD(0)		–	CE-Q
FP		TD(0)		Fictitious Play	Opponent Modelling / JALs

LP: linear programming
QP: quadratic programming
FP: fictitious play
CE: correlated equilibria solver

Table 3.8: Summary of algorithms for solving and learning in stochastic games. Each contains a matrix game solving component and a temporal differencing component. “Solving” algorithms assume the transition and reward function are known. “Learning” algorithms only receive observations of the transition and reward function.

3.3 Limitations

It is commonly recognized that humans are not capable of being fully rational. There is a whole body of work in the social sciences relating to understanding humans’ *bounded rationality* (Simon, 1982; Rubinstein, 1998). Bounded rationality is also a reality for artificial agents. In realistic problems agent limitations are unavoidable. These limitations may be due to the physical challenges of broken actuators, poor control of motion, or incomplete or noisy sensing. The limitations may also be due to the use of approximation techniques mentioned in Section 2.2.3. The effect is that agents are often incapable of optimal decision making.

In this section we focus on work examining limitations in multiagent settings. In single agent settings, it is possible to solve the bounded rationality problem through including the agents reasoning constraints into the optimization problem (Russell, 1997). Essentially, this is expecting an intelligent agent to do the best it can with its limitations. Limitations have far more implications in multiagent environments, such as stochastic games, since a common assumption in these settings is the rationality of the other agents.

We first examine work on defining a rational solution concept in situations where agents have computational limitations. We, then, examine work on responding to limitations in other agents’ behavior. We follow this with a brief summary of how this work relates to our own examination of the problem of multiagent learning in the presence of limited agents.

3.3.1 Responding to Computational Limitations

Larson and Sandholm (2001) consider the problem of reasoning in the presence of agents with limited computational resources. Specifically, they examine a two agent setting where agents have intractable individual problems, but may gain an improvement on their solutions by solving the also intractable joint problem. An agent's strategy determines how to invest its limited computational resources to improve its solution for its own problem, its opponent's problem, or the joint problem. When the deadline is reached, the agents must decide whether to simply use its own computed solution or bargain over a joint solution. Among other results, they demonstrate that the concept of Nash equilibrium can be extended to this setting, where the decision of computation is part of the agent's strategy. They call this concept *deliberation equilibria* and present algorithms for finding these equilibria solutions in certain situations.

The concept of deliberation equilibria has many similarities to the *restricted equilibria* concept we present in Chapter 5. The core idea is to incorporate the agents' limitations into their choices of strategies. Restricted equilibria, though, incorporate limitations as restrictions on the agents' choices, rather than incorporating limitations by adding computational decisions into their choices. This difference, though, is a fundamental one as the two approaches solve different problems. The complexity of decision-making in the Larson and Sandholm setting is directly due to the computational limitations placed on the agents. In Chapter 5, the complexity of decision-making is part of the task, and agent limitations are required in order to simplify the problem and make learning tractable. This core difference also explains the difference in results, since restricted equilibria are not guaranteed to exist in general.

3.3.2 Responding to Behavior Limitations

Another approach to handling to limitations is to respond to limitations in the other agents' policy or behavior. We examine three results in trying to exploit, or more generally account for, opponent behavior.

Game-tree Search with Opponent Knowledge Jansen's Ph.D. thesis (1992) examined the issue of whether "optimal" play in the game-theoretic sense is in fact always "optimal". In the case of a fallible opponent that may make an incorrect evaluation of future states, the answer is naturally no. He proposed the idea of *speculative play*, which uses a prediction function for the opponent in determining its opponent's moves when searching a game-tree. The thesis focuses on deterministic, alternating-move, perfect information games, specifically a subgame of chess. He demonstrated that using speculative play with end-game databases as knowledge of the opponent has the potential for improving play against humans. In this work, limitations on human rationality are recognized, and so game-theoretic optimal play is discarded for optimization against particulars of human play.

Planning with Opponent Models. Riley and Veloso (Riley & Veloso, 2000, 2002) present similar ideas for *using opponent models to make decisions* in a less abstract domain. They examine the problem of adapting to a specific opponent in simulated robotic soccer (Noda, Matsubara, Hiraki, & Frank, 1998). This work recognizes that modelling opponents is not a new idea and there has been a great deal of related research in the field of plan recognition. This work does contribute an actual technique for using opponent models to improve the team behavior, hence learning.

The team designer provides a set of opponent models of possible behavior of the other team. These models define a distribution of players' positions as a function of the ball's trajectory and their initial positions. During play, the actual opponent is matched to one of these models through Bayesian updating from actual observations. The model is then used to plan during set-play situations, such as throw-ins or goal-kicks. The planning uses hill-climbing on the ball-trajectory to find a trajectory with low probability of being intercepted by the other team *given the matched opponent model of their movement.* The planner then determines the positions of players and their actions so as to carry out the planned ball trajectory. This is compiled into a simple temporal network which can be executed in a distributed manner by the players. They demonstrate through controlled empirical experiments that the plans generated depend on the opponent model that is used during planning. The effectiveness of this technique in actual play against a real opponents, that may or may not match well with the models, is still unknown.

This technique allows a team of agents to adapt their behavior in a principled manner to better address a specific team. This does not strictly address the problem of concurrent learning in a multiagent setting. The critical assumption is that the opponents' play is not changing over time. The authors mention this briefly and provide a small change to their Bayesian update through the addition weight sharing. A small weight is added to each model to keep its probability bounded from zero. This effectively introduces a non-zero probability that the other team may change its behavior after each observation. There are no results, though, of the outcome of this approach if the other agents are also adapting. For example, it's not clear how robust the algorithm is to the other team employing a similar or identical approach to defending against a set-play.

Limitations as Finite Automata. Carmel and Markovitch (1996, 1997) examined a model-based approach to multiagent learning. They introduced algorithms for efficiently learning a model of the opponent as a deterministic finite automaton (DFA). Using this learned model, they then compute the optimal best response automaton. They examined this technique in the situation of repeated games and showed that their technique can learn effectively against a variety of DFA opponents. They also showed that their algorithm learned much more quickly (i.e., was more data efficient) than Q-learning, a model-free reinforcement learner. It's unknown how this compares with a model-based reinforcement learner such as prioritized sweeping (Moore & Atkeson, 1993).

This technique, though, does not entirely address the problem of concurrent learning in a multiagent environment. Although a learning rule can be be modeled by a DFA, the number of states that a rule requires may be huge. For example, Q-learning with a decaying learning rate would be a completely intractable DFA. Also, the model-based learning agent is required to be strictly more powerful than its opponent, since its learning rule both learns the opponent's DFA and also generates its own. It's unclear whether any sort of fixed point would be reached or even exists if two model-based DFA learners played each other. It's also not certain how DFA modelling scales to multiple state stochastic games.

All of these approaches to handling behavior limitations assume an asymmetric superiority over the other decision making agents. In the case of constructing DFA models of opponent behavior, there is an implicit assumption that the opponent behavior is simple enough to be constructed by the adapting agent. In the case of planning with opponent models, it is assumed that the opponent's behavior is not changing and so the learned model is accurate in predicting their future behavior. Although the work recognizes that other agents have limitations, they do so in a way that expects other agents to be strictly inferior. There are no results that apply these model-based techniques to situations of self-play, where all agents use the same adaptation procedure. Larson and Sanholm's work on computational limitations allows for limitations on the part of both agents. They do so by adding the complexity of computational decisions into the strategy space. Hence, their approach does not make this assumption.

Our approach in Chapters 5 and 6 examines learning in the context of limitations on behavior. However, we do not explicitly assume asymmetric superiority. Most of our results focus on self-play to insure that these algorithms are robust to equally capable opponents and teammates that are also learning.

3.4 Summary

In this Chapter we reviewed the work on solving stochastic games as well as learning in stochastic games. Although, these broad areas have different goals, we note that the algorithms have considerable similarities. For learning in stochastic games, we divided algorithms into two main types: equilibrium learners and best-response learners. These two categories are both useful for distinguishing the goals of an algorithm but also are useful in identifying the weaknesses of these approaches. We look at this in Chapter 4.

We also examined work on agent limitations, specifically research that seeks to account for limitations in making decisions. The most common assumption in this work is that agent limitations are the problem of the other agent, and should be exploited. They rarely consider the robustness of their techniques in symmetric situations where both agents are equally capable or limited. We return to limitations, their affect on games, and learning in Chapters 5 and 6.

Chapter 4

WoLF: Rational and Convergent Learning

In this chapter we examine the problem of learning in multiagent environments when agents are not limited. This chapter focuses on small and comparatively simple domains where approximation techniques, and therefore limitations, are not needed for learning.

We begin by introducing two desirable properties of learning algorithms, noticing how previous algorithms do not simultaneously achieve these properties. We then introduce the concept of a *variable learning rate* using the *WoLF*, "Win or Learn Fast", principle. In Section 4.2 we analyze theoretically the effect of this technique in a subclass of matrix games. We prove that a WoLF variable learning rate causes a non-convergent rational algorithm to converge to an equilibrium in self-play. In Section 4.3 we present a practical algorithm for learning in stochastic games. We empirically analyze this algorithm in Section 4.4, presenting results of its application in a variety of stochastic game domains, demonstrating it as a rational and convergent learning algorithm.

4.1 Properties

In this section we present two properties that are desirable for multiagent learning algorithms. We then examine how well previous algorithms have achieved these properties. Intuitively, the properties formalize the idea that a learner should learn a best-response when possible. Also, the learner should have some guarantee of convergence. Both of these properties' formal definitions rely on a notion of convergence which we define in the usual manner,

Definition 7 *Let $\pi_t(s_t, a)$ be player i's probability distribution over its actions at time t. A player's policy is said to converge to π_* if and only if for any $\epsilon > 0$ there exists a $T > 0$ such that,*

$$\forall t > T \; \forall a \in \mathcal{A}_i \qquad |\pi_t(s_t, a) - \pi_*(s_t, a)| < \epsilon.$$

A player's policy is said to converge to a stationary policy if and only if the player's policy converges to π_* for some $\pi_* \in \Pi_i$.

We can now define the two desirable properties of a learning algorithm: rationality and convergence.

Property 1 *(Rationality) If the other players' policies converge to stationary policies then the learning algorithm will converge to a policy that is a best-response to the other players' policies.*

This is a fairly basic property requiring the learner to learn and play a best-response policy when the other players play stationary policies, in which case a best-response policy does indeed exist. As we observed earlier, when other players play stationary policies, the stochastic game simply becomes an MDP. So, this property requires that the algorithm finds an optimal policy to this MDP.

Property 2 *(Convergence) The learner will necessarily converge to a stationary policy. This property will usually be conditioned on the other agents using an algorithm from some class of learning algorithms.*

This convergence property requires that, against some class of other players' learning algorithms (ideally a class encompassing most "useful" algorithms), the learner's policy will converge. For example, one might refer to convergence with respect to players with stationary policies, or convergence with respect to rational players.

In this work, we mostly focus on convergence in the case of self-play. That is, if all the players use the same learning algorithm, do the players' policies converge? This is a crucial and difficult step toward convergence against more general classes of players. In addition, ignoring the possibility of self-play makes the naïve assumption that other players are in some way inferior since they cannot be using an identical algorithm.

4.1.1 Relationship to Equilibria

Although these properties do not explicitly relate to notions of equilibria, there is still a strong relationship. If all of the players use rational learning algorithms and their policies converge, they must have converged to an equilibrium. This can be seen by considering one of the rational players: since the other players converge to a stationary policy, it will converge to a best-response because it is rational. Since this is true for all the players, then the resulting policies must be an equilibrium (i.e., each player is playing a best response to the other players). So, equilibria are fixed points of rational learning algorithms. In addition, if all the players use the same learning algorithm and it's rational and convergent in self-play, then the players are guaranteed to converge to a Nash equilibrium. This is because the convergent property guarantees convergence, and by the previous result, if they converge, it must be to an equilibrium.

4.1.2 Previous Algorithms

In Chapter 3 we presented a number of previous algorithms for learning in matrix games and stochastic games. We will now examine three representative algorithms from that presentation examining whether they achieve these properties. The first two are representative of best-response learners, and the last is representative of equilibrium learners.

Q-Learning. Q-learning's application to stochastic games simply ignores the existence of other agents, assuming its rewards and transitions are Markov and stationary. As a result, Q-learning traditionally learns deterministic policies. This naive approach does satisfy one of the two properties. If the other agents play, or converge to, stationary strategies then their Markovian assumption holds and they converge to an optimal response. So, Q-learning is rational. On the other hand, it is not generally convergent with other players that are also rational. This is obvious in games where the equilibrium is a mixed policy, which Q-learning cannot learn.

There are situations for which Q-learning is known to converge in self-play, such as iterated dominance solvable games and team games. As discussed in the context of fictitious play, iterated dominance solvable games are ones where the repeated process of removing strictly dominated actions from the game, leaves a single strategy for both players. Q-learning will eventually learn not to play the inferior action (although, possibly selecting it with a decreasing exploratory probability). Once the action ceases to be played often, that actions' rewards no longer affect the other player's Q-values will diminish, causing another action to be dominated for some player. This iteratively continues until the single action remains, for which Q-learning will play. Q-learning has also been explored in the context of team matrix games under the name independent learners (ILs) (Claus & Boutilier, 1998). They demonstrated that ILs will converge to a Nash equilibrium, although not necessarily a globally optimal equilibrium.

In addition to iterated dominance solvable games, there are other examples of games where Q-learning converges to an equilibrium in self-play, e.g., the grid navigation game from Section 2.4.1 (Bowling, 2000). The exact dividing line for convergence between games with only mixed equilibria and games that are iterated dominance solvable is still an open problem.

Opponent Modelling Q-Learning. Opponent modelling Q-learning or joint-action learning observes the actions of the other agents. It assumes that the other players are selecting actions based on a stationary policy, which it learns and plays optimally with respect to. Like Q-learning, it is rational, but not convergent, since it cannot learn mixed policies. Also like Q-learning, opponent modelling Q-learning will converge to an equilibrium in iterated dominance solvable games. The proof of this result is similar to the analysis of the model-based version, fictitious play (Fudenberg & Levine, 1999). In addition, joint-action learners have been demonstrated to be convergent in the setting of team matrix games, for which they were originally designed (Claus & Boutilier, 1998).

Minimax-Q. Minimax-Q takes a different approach. This algorithm observes both the actions and rewards of the other players and tries to learn a Nash equilibrium explicitly. The algorithm learns and plays the equilibrium independent of the behavior of other players. These algorithms are convergent with rational players, since they always converge to a stationary policy, so the other players being rational will converge to optimal responses. However, these algorithms are not rational. This is most obvious when considering a game of rock-paper-scissors against an opponent that almost always plays "Rock". Minimax-Q will still converge to the equilibrium solution, which is not optimal given the opponent's policy.

In essence, best-response learners are by definition rational, but they they are not in general convergent. On the other hand, equilibrium learners converge but are not rational, since they do so independently of the play of the other agents. For completeness' sake, this holds for regret minimizing algorithms (see (Jafari, Greenwald, Gondek, & Ercal, 2001) for empirical convergence failures), and gradient ascent algorithms, which we discuss next.

4.2 Theoretical Analysis

In this section we begin by examining gradient ascent as a technique for learning in simple two-player, two-action, general-sum repeated matrix games. We look at a theoretical analysis of this algorithm, observing that the algorithm can fail to converge. We follow by introducing the concept of a variable learning rate, and prove that this concept, in fact, causes gradient ascent to converge.

4.2.1 Gradient Ascent

Singh, Kearns, and Mansour (Singh et al., 2000b) examined the dynamics of using gradient ascent in two-player, two-action, repeated games. We can represent this problem as two matrices,

$$R_r = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \quad R_c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}.$$

Each player selects an action from the set $\{1, 2\}$ which determines the rewards or payoffs to the players. If the *row* player selects action i and the *column* player selects action j, then the row player receives the payoff r_{ij} and the column player receives the payoff c_{ij}.

Since this is a two-action game, a strategy (i.e., a probability distribution over the two available actions) can be represented as a single value. Let $\alpha \in [0, 1]$ be a strategy for the row player, where α corresponds to the probability that the player selects the first action and $1 - \alpha$ is the probability that the player selects the second action. Similarly, let β be a strategy for the column player. We can consider the joint strategy (α, β) as a point in $\mathbb{R}^2$ constrained to the unit square.

For any pair of strategies (α, β), we can write the expected payoffs that the row and column player will receive. Let $V_r(\alpha, \beta)$ and $V_c(\alpha, \beta)$ be these expected payoffs, respectively. So,

$$\begin{aligned} V_r(\alpha, \beta) &= \alpha\beta r_{11} + \alpha(1-\beta)r_{12} + (1-\alpha)\beta r_{21} + (1-\alpha)(1-\beta)r_{22} \\ &= u\alpha\beta + \alpha(r_{12} - r_{22}) + \beta(r_{21} - r_{22}) + r_{22} \end{aligned} \tag{4.1}$$

$$\begin{aligned} V_c(\alpha, \beta) &= \alpha\beta c_{11} + \alpha(1-\beta)c_{12} + (1-\alpha)\beta c_{21} + (1-\alpha)(1-\beta)c_{22} \\ &= u'\alpha\beta + \alpha(c_{12} - c_{22}) + \beta(c_{21} - c_{22}) + c_{22} \end{aligned} \tag{4.2}$$

where,

$$\begin{aligned} u &= r_{11} - r_{12} - r_{21} + r_{22} \\ u' &= c_{11} - c_{12} - c_{21} + c_{22}. \end{aligned}$$

A player can now consider the effect of changing its strategy on its expected payoff given the other player's strategy doesn't change. This can be computed as simply the partial derivative of its expected payoff with respect to its strategy,

$$\frac{\partial V_r(\alpha, \beta)}{\partial \alpha} = \beta u + (r_{12} - r_{22}) \tag{4.3}$$

$$\frac{\partial V_c(\alpha, \beta)}{\partial \beta} = \alpha u' + (c_{21} - c_{22}). \tag{4.4}$$

In the gradient ascent algorithm a player adjusts its strategy after each iteration so as to increase its expected payoffs. This means that the player moves its strategy in the direction of the current gradient with some step size, η. If (α_k, β_k) are the strategies on the kth iteration, and both players are using gradient ascent then the new strategies are,

$$\begin{aligned}\alpha_{k+1} &= \alpha_k + \eta \frac{\partial V_r(\alpha_k, \beta_k)}{\partial \alpha_k} \\ \beta_{k+1} &= \beta_k + \eta \frac{\partial V_r(\alpha_k, \beta_k)}{\partial \beta_k}.\end{aligned}$$

If the gradient moves the strategy out of the valid probability space (i.e., the range [0, 1]) then the gradient is forced to be zero. When considering the strategy pair in tandem, this amounts to projecting the joint gradient onto the unit square for points on this boundary with a gradient pointing outside of the boundary. The question to consider, then, is what can we expect to happen if both players are using gradient ascent to update their strategies.

Notice that this algorithm is *rational* by the properties defined in Section 4.1. This is because fixing the other player's strategy causes the player's gradient to become constant, and eventually forces the player to converge to the optimal pure strategy response, or remain with its current strategy if it already is a best-response. On the other hand the algorithm is not *convergent*, which is shown in (Singh et al., 2000b). This is despite the fact that the algorithm can and does play mixed strategies.

The analysis, by Singh and colleagues, of gradient ascent examines the dynamics of the learners in the case of an infinitesimal step size ($\lim_{\eta \to 0}$). They call this algorithm Infinitesimal Gradient Ascent (IGA). They observe that an algorithm with an appropriately decreasing step size will approach the dynamics of IGA, making the analysis of the infinitesimal case of particular interest. In the next section we briefly outline their analysis.

4.2.2 Analysis of IGA

The main conclusion of Singh, Kearns, and Mansour (2000b) is the following theorem.

Theorem 1 *If both players follow Infinitesimal Gradient Ascent (IGA), where $\eta \to 0$, then their strategies converge to a Nash equilibrium* OR *the average payoffs over time converge in the limit to the expected payoffs of a Nash equilibrium.*

Their proof of this theorem proceeds by examining the dynamics of the strategy pair, (α, β). This is an affine dynamical system in $\mathbb{R}^2$ where the dynamics are defined by the differential equation,

$$\begin{bmatrix} \frac{\partial \alpha}{\partial t} \\ \frac{\partial \beta}{\partial t} \end{bmatrix} = \begin{bmatrix} 0 & u \\ u' & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} + \begin{bmatrix} (r_{12} - r_{22}) \\ (c_{21} - c_{22}) \end{bmatrix}.$$

If we define U to be the multiplicative matrix term above with off-diagonal values u and u', then we can classify the dynamics of the system based on properties of U. From dynamical systems theory (Reinhard, 1987), if U is invertible then there are only two qualitative forms for the dynamics of the system, depending on whether U has purely real or purely imaginary eigenvalues. This results in three cases: U is not invertible, U has purely real eigenvalues, or U has purely imaginary eigenvalues. The qualitative forms of these different cases are shown in Figure 4.1. Their analysis then proceeded by examining each case geometrically. One important consideration is that the basic forms above are for the unconstrained dynamics, not the dynamics that projects the gradient onto the unit square. Basically, this requires considering all possible positions of the unit square relative to the dynamics shown in Figure 4.1.

One crucial aspect of their analysis is the consideration of points with zero gradient in the constrained dynamics, which they show to correspond to Nash equilibria. This is also discussed in Lemma 2. In the unconstrained dynamics, there exists at most one point with zero gradient, which is called the center and denoted (α^*, β^*). This point can be found mathematically by setting Equations 4.3 and 4.4 to zero and solving to obtain,

$$(\alpha^*, \beta^*) = \left(\frac{(c_{22} - c_{21})}{u'}, \frac{(r_{22} - r_{12})}{u} \right).$$

Notice that the center may not even be inside the unit square. In addition, if U is not invertible then there is no point of zero gradient in the unconstrained dynamics. But in the constrained dynamics, where gradients on the boundaries of the unit square are projected onto the unit square, additional points of zero gradient may exist. When IGA converges it is to one of these points with zero gradient.

This theorem is an exciting result since it is one of the first convergence results for a rational multiagent learning algorithm. The notion of convergence, though, is rather weak. In fact, not only may the players' policies not converge when playing gradient ascent but the expected payoffs may not converge either. Furthermore, at any moment in time the expected payoff of a player could be arbitrarily poor.[1] Not only does this make it difficult to evaluate a learner, it also could be potentially disastrous when applied with temporal differencing for multiple state stochastic games. A cycling value of a state is a further violation of the stationary assumption upon which temporal differencing depends. The danger is that if the delayed reward takes too much time to propagate to the state with the responsible decision, then the reward for finally making the correct decision could be lost if the state's value has cycled into its period of low reward.

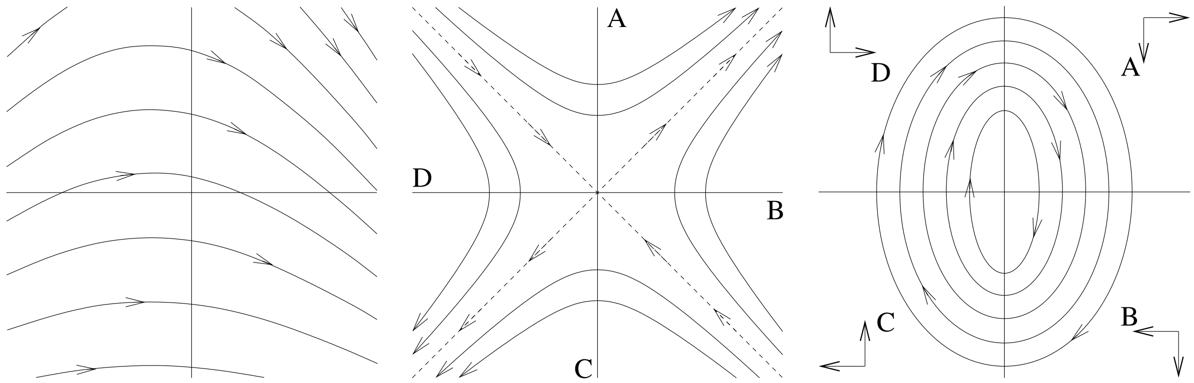
In the next section we examine a method for addressing this convergence problem. We then prove that this new method has the stronger notion of convergence, i.e., players' strategies *always* converge to a Nash equilibrium.

4.2.3 Variable Learning Rate

We now introduce the concept and study the impact of a variable learning rate. In the gradient ascent algorithm presented above, the steps taken in the direction of the gradient were constant. We now allow them to vary over time, thus changing the update rules to,

$$\begin{aligned} \alpha_{k+1} &= \alpha_k + \eta \ell_k^r \frac{\partial V_r(\alpha_k, \beta_k)}{\partial \alpha} \\ \beta_{k+1} &= \beta_k + \eta \ell_k^c \frac{\partial V_r(\alpha_k, \beta_k)}{\partial \beta} \end{aligned}$$

[1] The idea that average payoffs converge only means that if there's a period of arbitrarily low payoffs there must be some corresponding period in the past or in the future of arbitrarily high payoffs.



(a) U is not invertible (b) U has real eigenvalues (c) U has imaginary eigenvalues

Figure 4.1: Qualitative forms of the IGA dynamics.

where,

$$\ell_k^r, \ell_k^c \in [\ell_{\min}, \ell_{\max}] > 0.$$

At the kth iteration the algorithm takes a step of size $\eta\ell_k$ in the direction of the gradient. Notice the restrictions on ℓ_k require that it be strictly positive and bounded, thus bounding the step sizes as well.

The specific method for varying the learning rate that we contribute is the WoLF ("Win or Learn Fast") principle. The essence of this method is to learn quickly when losing, and cautiously when winning. The intuition is that a learner should adapt quickly when it is doing more poorly than expected. When it is doing better than expected, it should be cautious since the other players are likely to change their policy. The heart of the algorithm is how to determine whether a player is winning or losing. For the analysis in this section each player selects a Nash equilibrium and compares their expected payoff with the payoff they would receive if they played according to the selected equilibrium strategy. Let α^e be the equilibrium strategy selected by the row player, and β^e be the equilibrium strategy selected by the column player. Notice that no requirement is made that the players choose the same equilibrium (i.e., the strategy pair (α^e, β^e) may not be a Nash equilibrium). Formally,

$$\ell_k^r = \begin{cases} \ell_{\min} & \text{if } V_r(\alpha_k, \beta_k) > V_r(\alpha^e, \beta_k) & \textsc{winning} \\ \ell_{\max} & \text{otherwise} & \textsc{losing} \end{cases}$$

$$\ell_k^c = \begin{cases} \ell_{\min} & \text{if } V_c(\alpha_k, \beta_k) > V_c(\alpha_k, \beta^e) & \textsc{winning} \\ \ell_{\max} & \text{otherwise} & \textsc{losing} \end{cases}$$

With a variable learning rate such as this we can still consider the case of an infinitesimal step size ($\lim_{\eta \to 0}$). We call this algorithm WoLF-IGA and in the next section show that the WoLF adjustment has a very interesting effect on the convergence of the algorithm.

4.2.4 Analysis of WoLF-IGA

We prove the following result.

Theorem 2 *If in a two-person, two-action repeated game, both players follow the WoLF-IGA algorithm (with $\ell_{\max} > \ell_{\min}$), then their strategies will converge to a Nash equilibrium.*

Notice that this is the more standard notion of convergence and strictly stronger than what is true for basic IGA.

The proof of this theorem follows closely with the proof of Theorem 1 from Singh and colleagues (Singh et al., 2000b), by examining the possible cases for the dynamics of the learners. First, let us write down the differential equations that define the system with an infinitesimal step size,

$$\begin{bmatrix} \frac{\partial \alpha}{\partial t} \\ \frac{\partial \beta}{\partial t} \end{bmatrix} = \begin{bmatrix} 0 & \ell^r(t)u \\ \ell^c(t)u' & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} + \begin{bmatrix} \ell^r(t)(r_{12} - r_{22}) \\ \ell^c(t)(c_{21} - c_{22}) \end{bmatrix}.$$

We call the multiplicative matrix with off-diagonal entries $U(t)$ since it now depends on the learning rates at time t, $\ell^r(t)$ and $\ell^c(t)$. At time t, the qualitative form of the dynamics is determined by the $U(t)$ matrix and can be summarized into three general cases,

- $U(t)$ is not invertible,
- $U(t)$ has purely real eigenvalues, or
- $U(t)$ has purely imaginary eigenvalues.

The first thing to note is that the above cases do not depend on t. The following lemma is even stronger.

Lemma 1 *$U(t)$ is invertible if and only if U (as defined for IGA in Section 4.2.2) is invertible. $U(t)$ has purely imaginary eigenvalues if and only if U has purely imaginary eigenvalues. $U(t)$ has purely real eigenvalues if and only if U has purely real eigenvalues.*

Proof. Since $\ell^{r,c}(t)$ is positive, it is trivial to see that $(\ell^r(t)u)(\ell^c(t)u') = (\ell^r(t)\ell^c(t))uu'$ is greater-than, less-than, or equal-to zero, if and only if uu' is greater-than, less-than, or equal-to zero, respectively. Since these are the exact conditions of invertibility and purely real/imaginary eigenvalues the lemma is true. $\square$

So $U(t)$ always satisfies the same case (and therefore has the same general dynamics) as IGA *without* a variable learning rate. In the sections that follow we examine each of these cases separately. The proofs of most of the cases proceed identically to the proof for IGA. In fact, most of the proofs do not rely on any particular learning rate adjustment at all. Only in the final sub-case of the final case are we forced to deviate from their arguments. This is due to the fact that variable learning rates, in general, do not change the overall direction of the gradient (i.e., the sign of the partial derivatives). Since most of the proof of IGA's convergence only depends on the signs of the derivatives, we can use the same arguments. For these cases we present only an abbreviated proof of convergence to illustrate that the variable learning rate does not affect their arguments. We recommend the IGA analysis (Singh et al., 2000b) for a more thorough examination including helpful diagrams. In the remaining sub-case, where IGA is shown not to converge, we show that in this case WoLF-IGA converges to a Nash equilibrium.

We make liberal use of a crucial lemma from their proof for IGA. This lemma implies that if the algorithms converge then what the strategies converge to must be a Nash equilibrium.

Lemma 2 *If, in following IGA or WoLF-IGA,* $\lim_{t\to\infty}(\alpha(t), \beta(t)) = (\alpha_c, \beta_c)$*, then* (α_c, β_c) *is a Nash equilibrium.*

Proof. The proof for IGA is given in (Singh et al., 2000b), and shows that the algorithm converges if and only if the projected gradient is zero, and such strategy pairs must be a Nash equilibrium. For WoLF-IGA notice also that the algorithm converges if and only if the projected gradient is zero, which is true if and only if the projected gradient in IGA is zero. Therefore that point must be a Nash equilibrium. □

Now we examine the individual cases.

$U(t)$ is Not Invertible

In this case the dynamics of the strategy pair has the qualitative form shown in Figure 4.1(a).

Lemma 3 *When U(t) is not invertible, IGA with any learning rate adjustment leads the strategy pair to converge to a point on the boundary that is a Nash equilibrium.*

Proof. Notice that $U(t)$ is not invertible if and only if u or u' is zero. Without loss of generality, assume u is zero, then the gradient for the column player is constant. The column player's strategy, β, converges to either zero or one (depending on whether the gradient was positive or negative). At this point, the row player's gradient becomes constant and therefore must also converge to zero or one, depending on the sign of the gradient. The joint strategy therefore converges to some corner, which by Lemma 2 is a Nash equilibrium. □

$U(t)$ has Real Eigenvalues

In this case the dynamics of the strategy pair has the qualitative form shown in Figure 4.1(b).

Lemma 4 *When U(t) has real eigenvalues, IGA with any learning rate adjustment leads the strategy pair to converge to a point that is a Nash equilibrium.*

Proof. Without loss of generality, assume that $u, u' > 0$. This is the dynamics show in Figure 4.1(b). Consider the case where the center is inside the unit square. Notice that if the strategy pair is in quadrant A relative to the center, the gradient is always up and right. Therefore, any strategy pair in this region eventually converges to the upper-right corner of the unit square. Likewise, strategies in quadrant C always converge to the bottom-left corner. Now consider a strategy pair in quadrant B. The gradient is always up and left, and therefore the strategy eventually exits this quadrant, entering quadrant A or C, or possibly hitting the center. At the center the gradient is zero, and so it has converged. If it enters quadrants A or C then we've already shown it converges to the upper-right or lower-left corner. Therefore, the strategies always converge and by Lemma 2 the point must be a Nash equilibrium. Cases where the center is not within the unit square or is on the boundary of the unit square can also be shown to converge by a similar analysis, and are discussed in (Singh et al., 2000b). □

$U(t)$ has Imaginary Eigenvalues

In this case the dynamics of the strategy pair has the qualitative form shown in Figure 4.1(c). This case can be further broken down into sub-cases depending where the unit square is in relation to the center.

Center is Not Inside the Unit Square. In this case we still can use the same argument as for IGA.

Lemma 5 *When U(t) has imaginary eigenvalues and the center, (α^*, β^*), is not inside the unit square, IGA with any learning rate adjustment leads the strategy pair to converge to a point on the boundary that is a Nash equilibrium.*

Proof. There are three cases to consider. The first is the unit square lies entirely within a single quadrant. In this case the direction of the gradient is constant (e.g., down-and-right in quadrant A). Therefore, the strategies converge to one of these corners (e.g., bottom-right corner in quadrant A). The second case is the unit square is entirely within two neighboring quadrants. Consider the case that it lies entirely within quadrants A and D. The gradient always points to the right and therefore the strategy eventually hits the right boundary at which point it is in quadrant A and the gradient is pointing downward. Therefore, in this case it converges to the bottom right corner. We can similarly show convergence for other pairs of quadrants. The third and final case is when the center is on the boundary of the unit square. In this case some points along the boundary have a projected gradient of zero. By similar arguments to those above, any strategy converges to one of these boundary points. See (Singh et al., 2000b) for a diagram and further explanation. Since in all cases the strategy pairs converge, by Lemma 2 they must have converged to a Nash equilibrium. □

Center is Inside the Unit Square. This is the final sub-case and is the point where the dynamics of IGA and WoLF-IGA qualitatively differ. We show that, although IGA does not converge in this case, WoLF-IGA does. The proof identifies the areas of the strategy space where the players are "winning" and "losing" and shows that the trajectories are actually piecewise elliptical in such a way that they spiral toward the center. All of the lemmas in this subsection implicitly assume that $U(t)$ has imaginary eigenvalues and the center is inside the unit square. We begin with the following lemma that considers the dynamics for fixed learning rates.

Lemma 6 *If the learning rates, ℓ^r and ℓ^c, remain constant, then the trajectory of the strategy pair is an elliptical orbit around the center, (α^*, β^*), and the axes of this ellipse are,*

$$\begin{bmatrix} 0 \\ \sqrt{\frac{\ell^c |u|}{\ell^r |u'|}} \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Proof. This is just a result from dynamical systems theory (Reinhard, 1987) as mentioned in (Singh et al., 2000b) when $U(t)$ has imaginary eigenvalues. □

We now need the critical lemma that identifies the areas of strategy space where the players are using a constant learning rate. Notice that this corresponds to the areas where the players are "winning" or "losing".

Lemma 7 *The player is "winning" if and only if that player's strategy is moving away from the center.*

Proof. Notice that in this sub-case where $U(t)$ has imaginary eigenvalues and the center is within the unit square, the game has a single Nash equilibrium, which is the center. So, the players' selected equilibrium strategies for the WoLF principle must be the center, i.e., $(\alpha^e, \beta^e) = (\alpha^*, \beta^*)$. Now, consider the row player. The player is "winning" when its current expected payoff is larger than the expected payoffs if it were to play its selected equilibrium. This can be written as,

$$V_r(\alpha, \beta) - V_r(\alpha^e, \beta) > 0. \tag{4.5}$$

We can rewrite the left hand side of Inequality 4.5 by using Equation 4.1,

$$\begin{array}{c}(\alpha\beta u + \alpha(r_{12} - r_{22}) + \beta(r_{21} - r_{22}) + r_{22}) - \\ (\alpha^e\beta u + \alpha^e(r_{12} - r_{22}) + \beta(r_{21} - r_{22}) + r_{22}).\end{array} \tag{4.6}$$

Using Expression 4.6 in Inequality 4.5, we substitute the center for the equilibrium strategies, and then simplify making use of equation 4.3, as presented in Section 4.2.1.

$$\begin{aligned}(\alpha - \alpha^*)\beta u + (\alpha - \alpha^*)(r_{12} - r_{22}) &> 0 && (4.7)\\ (\alpha - \alpha^*)(\beta u + (r_{12} - r_{22})) &> 0 && (4.8)\\ (\alpha - \alpha^*)\frac{\partial V_r(\alpha, \beta)}{\partial \alpha} &> 0. && (4.9)\end{aligned}$$

Notice that Inequality 4.9 is satisfied if and only if the two left hand factors have the same sign. This is true if and only if the player's strategy, α, is greater than the strategy at the center, α^*, and it is increasing, or it is smaller than the center and decreasing. So the player is winning if and only if its strategy is moving away from the center. The same can be shown for the column player. □

Corollary 1 *Throughout any one quadrant, the learning rate is constant.*

Combining Lemmas 6 and 7, we find that the trajectories are piece-wise elliptical orbits around the center, where the pieces correspond to the quadrants defined by the center. We can now prove convergence for a limited number of starting strategy pairs. We then use this lemma to prove convergence for any initial strategy pairs.

Lemma 8 *For any initial strategy pair, $(\alpha^*, \beta^* + \beta_0)$ or $(\alpha^* + \alpha_0, \beta^*)$, that is "sufficiently close" to the center, the strategy pair converges to the center. "Sufficiently close" here means that the elliptical trajectory from this point defined when both players use 1 as their learning rate lies entirely within the unit square.*

Proof. Without loss of generality assume $u > 0$ and $u' < 0$. This is the case shown in Figure 4.1(c). Let $l = \sqrt{\frac{\ell_{\min}}{\ell_{\max}}} < 1.0$, and $r = \sqrt{\frac{|u'|}{|u|}}$. Consider an initial strategy $(\alpha^*, \beta^* + \beta_0)$ with $\beta_0 > 0$.

From Lemma 6, for any fixed learning rates for the players, the trajectory forms an ellipse centered at (α^*, β^*) and with the ratio of its y-radius to its x-radius equal to,

$$\sqrt{\frac{\ell^c}{\ell^r}}r.$$

Since the trajectory is piecewise elliptical we can consider the ellipse that the trajectory follows while in each quadrant. This is shown graphically in Figure 4.2. As the trajectory travels through quadrant A, by Lemma 7, we can observe that the row player is "winning" and the column player is "losing". Therefore, $\ell^r = \ell_{\min}$ and $\ell^c = \ell_{\max}$, so the ratio of the ellipse's axes are r/l, and this ellipse crosses into quadrant B at the point $(\alpha^* + \beta_0 \frac{l}{r}, \beta^*)$. Similarly, in quadrant B, the row player is "losing" and the column player is "winning" therefore the ratio of the ellipse's axes are rl and the ellipse crosses into quadrant C at the point $(\alpha^*, \beta^* - \beta_0 l^2)$.

We can continue this to return to the axis where the trajectory began. The strategy pair at that point is $(\alpha^*, \beta^* + \beta_0 l^4)$. So, for each orbit around the center we decrease the distance to the center by a factor of $l^4 < 1.0$, and therefore the trajectory converges to the center. We can reason identically for any other sufficiently close initial strategies on the axes. □

Lemma 9 *When U(t) has imaginary eigenvalues and the center, (α^*, β^*), is inside the unit square, WoLF-IGA leads the strategy pair to converge to the center, and therefore to a Nash equilibrium.*

Proof. The proof just involves the application of Lemma 8. Consider the largest ellipse, when both players' learning rates are one, that fits entirely within the unit square. This ellipse touches the boundary of the unit square and does so at the boundary of two quadrants. Now consider any initial strategy pair. The strategy pair follows piecewise elliptical orbits or moves along the unit square boundary while "circling" the center, that is traveling through the four quadrants in a clockwise or counter-clockwise fashion. At some point it must cross the boundary between the same two quadrants mentioned above. At this point it is on or inside the largest ellipse defined when players have a learning rate of one. Therefore we can apply Lemma 8 and so the trajectory converges to the center. So, from any initial strategy pair the trajectory converges to the center, which is a Nash equilibrium. □

Lemmas 3, 4, 5, and 9 combine to prove Theorem 2. In summary, the WoLF principle strengthens the IGA convergence result. In self-play with WoLF-IGA, players' strategies and their expected payoffs converge to Nash equilibrium strategies and payoffs of the matrix game. This contributes a reinforcement-based algorithm that is provably *rational* and *convergent* (in self-play) for this restricted class of repeated games. This result can be generalized beyond self-play in the following corollary.

Corollary 2 *If in a two-person, two-action repeated game, both players follow the WoLF-IGA algorithm but with different $\ell_{\min}$ and $\ell_{\max}$, then their strategies converge to a Nash equilibrium if,*

$$\frac{\ell^r_{\min}\,\ell^c_{\min}}{\ell^r_{\max}\,\ell^c_{\max}} < 1.$$

Specifically, WoLF-IGA (with $\ell_{\max} > \ell_{\min}$) versus IGA ($\ell_{\max} = \ell_{\min}$) converges to a Nash equilibrium.

Proof. The proof is almost identical to Theorem 2. The only deviation is for the imaginary eigenvalue case where the center is inside the unit square. In this case the proof of Lemma 8 is amended.

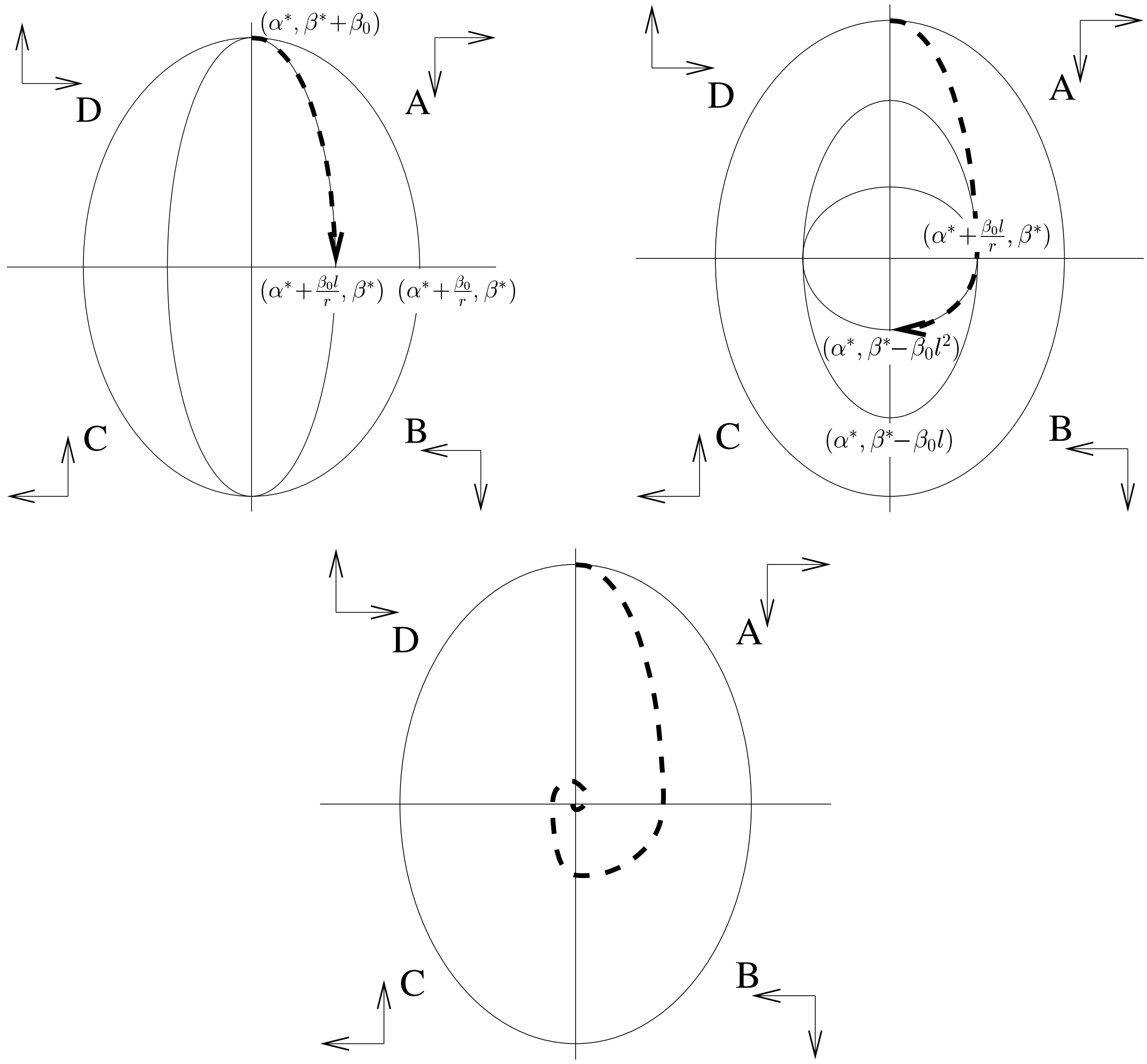


Figure 4.2: The trajectory of strategies using WoLF-IGA when $U(t)$ has imaginary eigenvalues and the center is inside the unit square.

Let $l^r = \sqrt{\frac{\ell^r_{\min}}{\ell^c_{\max}}}$ and $l^c = \sqrt{\frac{\ell^c_{\min}}{\ell^r_{\max}}}$. Following the same argument of the trajectory through the quadrants, after a revolution around the center the new position is $(\alpha^*, \beta^* + \beta_0(l^r l^c)^2)$. Since,

$$(l^r l^c)^2 = \frac{\ell^r_{\min}\,\ell^c_{\min}}{\ell^r_{\max}\,\ell^c_{\max}} < 1,$$

the trajectory converges to the center. The remainder of the proof is identical to that of Theorem 2. □

We return to examining WoLF outside of self-play in Section 4.4.5, where we examine this situation empirically in a more complex domain.

4.2.5 Discussion

There are some final points to be made about this result. First, we present some further justification for the WoLF principle as it has been used in other learning related problems. Second, we present a short discussion on determining when a player is "winning". Finally, we look at the knowledge the WoLF-IGA algorithm requires. These requirements are later relaxed in a more practical algorithm in Section 4.3.

Why WoLF?

Apart from this theoretical result, the WoLF principle may appear to be just an unfounded heuristic. But actually it has been studied in some form in other areas, notably when considering an adversary. In evolutionary game theory the *adjusted replicator dynamics* (Weibull, 1995) scales the individual's growth rate by the inverse of the overall success of the population. This causes the population's composition to change more quickly when the population as a whole is performing poorly. A form of this also appears as a modification to the *randomized weighted majority* algorithm (Blum & Burch, 1997). In this algorithm, when an expert makes a mistake, a portion of its weight loss is redistributed among the other experts. If the algorithm is placing large weights on mistaken experts (i.e., the algorithm is "losing"), then a larger portion of the weights are redistributed (i.e., the algorithm adapts more quickly).

More generally, one can see a similar principle at work in worst-case analysis of amortized algorithms. For example, consider splay trees (Sleator & Tarjan, 1985), which are structures for maintaining a balanced binary search tree on-line. The algorithm performs a small number of tree rotations, which balance the tree, if the item queried is near the root. If it is far from the root, then many tree rotations are performed. If one considers the number of tree rotations as the measure of the speed of "learning", and the depth of the queried node as the measure of "winning" or "losing", then this is another example use of something like the WoLF principle. Again, small changes are made when the algorithm is performing well, while large changes are made when it is performing poorly.

Defining "Winning"

The WoLF principle for adjusting the learning rate is to learn faster when losing and more slowly when winning. This places a great deal of emphasis on how to determine that a player is winning.

In the description of WoLF-IGA above, the row-player was considered winning when,

$$V_r(\alpha_k, \beta_k) > V_r(\alpha^e, \beta_k).$$

Essentially, a player is winning if it would prefer its current strategy to that of playing some equilibrium strategy against the other player's current strategy.

Another possible choice of determining when a player is winning is if its expected payoff is currently larger than the value of the game's equilibrium (or some equilibrium if multiple exist). If we consider the final subcase in Section 4.2.4, then mathematically this would correspond to,

$$V_r(\alpha_k, \beta_k) \quad > \quad V_r(\alpha^*, \beta^*). \tag{4.10}$$

It is interesting to note that in zero-sum games with mixed strategy equilibria these two rules are actually identical.

In general-sum games, though, this is not necessarily the case. This situation should bring to light the differences between the two methods. There exist general-sum, two-player, two-action games with points in the strategy space where the player is actually receiving a lower expected payoff than the equilibrium value, but not lower than the expected payoff that the equilibrium strategy would receive against the other player's strategy. Essentially, the player is not doing poorly because its strategy is poor, but rather because of the play of the other player. It is at this point that the gradient is likely to be moving the strategy away from the equilibrium, and therefore if Inequality 4.10 was used, the player would move away from the equilibrium quickly, discouraging convergence. [2]

Requirements

The gradient ascent and WoLF gradient ascent algorithms make some strict requirements on the knowledge that is available to the player. Specifically, basic gradient ascent requires the following to be known: the player's own payoff matrix (i.e., R_r for the row-player), and the actual distribution of actions the other player is playing (β_k for the row-player). These two requirements are both very strong, particularly the latter. Often the payoffs are not known and rather need to be learned via experience, and even more often only the action selected by the other player is known (if even that) but not the player's distribution over actions.

WoLF gradient ascent proceeds to add a further requirement that a Nash equilibrium must also be known. In this case, this is not really extra knowledge, since it can be computed from the known payoffs. If we want to make this algorithm more general, though, it needs to be addressed how the algorithm determines whether it's winning or losing when an equilibrium is not known because the payoffs are not known. In the next section, we look at an algorithm that removes all of these knowledge constraints and we give empirical results on more general stochastic games, not just two-player, two-action matrix games.

[2] An example of such a matrix game is, $R_r = \begin{bmatrix} 0 & 3 \\ 1 & 2 \end{bmatrix}$, $R_c = \begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix}$, with the strategy point, $(\alpha, \beta) = (0.1, 0.9)$. The only Nash equilibrium is $(0.5, 0.5)$. Hence, $V_r(\alpha^*, \beta) = 0.7 < V_r(\alpha, \beta) = 1.02 < V_r(\alpha^*, \beta^*) = 2$, and so the two rules would disagree. Notice the gradient, $\partial V_r(\alpha, \beta)/\partial \alpha = -0.8$, is negative causing α to decrease away from the equilibrium.

1. Let $\alpha \in (0,1]$ and $\delta \in (0,\infty]$ be learning rates. Initialize,
$$Q(s,a) \leftarrow 0, \qquad \pi(s,a) \leftarrow \frac{1}{|\mathcal{A}_i|}.$$
2. Repeat,
   (a) From state s select action a according to mixed strategy $\pi(s)$ with suitable exploration (see Footnote 3).
   (b) Observing reward r and next state s',
   $$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a')\right).$$
   (c) Step π closer to the optimal policy w.r.t. Q,
   $$\pi(s,a) \quad \leftarrow \quad \pi(s,a) + \Delta_{sa} \ ,$$
   while constrained to a legal probability distribution,
   $$\Delta_{sa} \quad = \quad \begin{cases} -\delta_{sa} & \text{if } a \neq \text{argmax}_{a'} Q(s,a') \\ \sum_{a' \neq a} \delta_{sa'} & \text{otherwise} \end{cases}$$
   $$\delta_{sa} \quad = \quad \min\left(\pi(s,a), \frac{\delta}{|\mathcal{A}_i| - 1}\right).$$

Table 4.1: Policy hill-climbing algorithm (PHC) for player i.

4.3 A Practical Algorithm

We now present a more general algorithm using the WoLF principle to vary the learning rate. We begin by describing a simple rational algorithm, policy hill-climbing (PHC), that does not converge. This algorithm is similar to gradient ascent, but does not require as much knowledge. We then describe WoLF-PHC, which varies the learning rate according to an approximate notion of winning. In Section 4.4, these algorithms are examined empirically in a number of domains from matrix games to zero-sum and general-sum stochastic games.

4.3.1 Policy Hill-Climbing

We first present a simple rational learning algorithm that is capable of playing mixed strategies. It is a simple extension of Q-learning and is shown in Table 4.1. The algorithm, in essence, performs hill-climbing in the space of mixed policies. Q-values are maintained just as in normal Q-learning. In addition the algorithm maintains the current mixed policy. The policy is improved by increasing the probability that it selects the highest valued action according to a learning rate $\delta \in (0,\infty]$. Notice that when $\delta = \infty$ the algorithm is equivalent to Q-learning, since with each step the policy moves to the greedy policy executing the highest valued action with probability 1 (modulo exploration).

This technique, like Q-learning, is rational and converges to an optimal policy if the other play-

1. Let $\alpha \in (0, 1]$, $\delta_l > \delta_w \in (0, 1]$ be learning rates. Initialize,
$$Q(s, a) \leftarrow 0, \qquad \pi(s, a) \leftarrow \frac{1}{|\mathcal{A}_i|}, \qquad C(s) \leftarrow 0.$$
2. Repeat,
   (a) From state s select action a according to mixed strategy $\pi(s)$ with suitable exploration (see Footnote 3).
   (b) Observing reward r and next state s',
$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') \right).$$
   (c) Update estimate of average policy, $\bar{\pi}$,
$$\begin{aligned} C(s) &\leftarrow C(s) + 1 \\ \forall a' \in \mathcal{A}_i \quad \bar{\pi}(s, a') &\leftarrow \bar{\pi}(s, a') + \frac{1}{C(s)} \left(\pi(s, a') - \bar{\pi}(s, a')\right). \end{aligned}$$
   (d) Step π closer to the optimal policy w.r.t. Q. Same as PHC in Table 4.1(c), but with
$$\delta = \begin{cases} \delta_w & \text{if } \sum_{a'} \pi(s, a')Q(s, a') > \sum_{a'} \bar{\pi}(s, a')Q(s, a') \\ \delta_l & \text{otherwise} \end{cases}.$$

Table 4.2: WoLF policy-hill climbing algorithm (WoLF-PHC) for player i.

ers are playing stationary strategies. The proof follows from the proof for Q-learning (Watkins, 1989), which guarantees the Q values converge to Q^* with a suitable exploration policy.[3] π converges to a policy that is greedy according to Q, which is converging to Q^*, and therefore converges to a best response. Despite the fact that it is rational and can play mixed policies, it still is not convergent. We show examples of its convergence failures in Section 4.4.

4.3.2 WoLF Policy Hill-Climbing

We now introduce a modification to the naïve policy hill-climbing algorithm that encourages the desired convergence property without sacrificing the rational property. The algorithm uses a variable learning rate with the WoLF, "Win or Learn Fast", principle. The full algorithm is shown in Table 4.2.

As a reminder, the WoLF principle results in the agent learning quickly when it is doing poorly and cautiously when it is performing well. This change in the learning rate aids in convergence by not overfitting to the other players' changing policies. There's an expectation that the other players' policies are likely to change in cases when things seem "too good to be true". Practically, the algorithm requires two learning parameters, $\delta_l > \delta_w$. The parameter that is used to update the policy depends on whether the agent is currently determined to be winning or losing. This determination is done by comparing whether the current expected value is greater than the current expected value

[3] We continue our assumption that an exploration policy suitable for online learning is used. See Section 2.2.3.

of playing the *average policy.* If the current expected value is lower (i.e., the agent is "losing"), then the larger learning rate, δ_l, is used, otherwise δ_w is used. The average policy is intended to take the place of the unknown equilibrium policy. For many games, averaging over greedy policies does, in fact, approximate the equilibrium (e.g., this is a property of fictitious play (Vrieze, 1987)). In summary, WoLF-PHC introduces (i) two learning rates, and (ii) the determination of winning and losing using the average policy as an approximation for the equilibrium policy.

WoLF policy hill-climbing is still rational, since only the speed of learning is altered. Its convergence properties, though, are quite different. In the next section we show examples that this technique converges to best-response policies for a number and variety of stochastic games.

4.4 Results

We now show results of applying policy hill-climbing and WoLF policy hill-climbing to a number of different games. The domains include two matrix games that help to show how the algorithms work and to demonstrate empirically that the WoLF variable learning rate is having the same effect as in the theoretical results in Section 4.2. The algorithms were also applied to two multiple state stochastic games. One is the general-sum grid navigation domain from Section 2.4.1, and the other is the zero-sum soccer game from Section 2.4.1. We also examine a matrix game that involves three players, instead of just two. Finally, we briefly look at the performance of WoLF in a few experiments where training was not against an identical algorithm.

The first set of experiments involves training the players using the same learning algorithm, i.e., self-play. Since PHC and WoLF-PHC are rational, we know that if they converge against themselves, then they must converge to a Nash equilibrium (see Section 4.1.1). For the matrix game experiments, $\delta_l/\delta_w = 2$, but for the stochastic game results a more aggressive $\delta_l/\delta_w = 4$ was used. The smaller ratio was used in the matrix games to *slow down* the convergence of the WoLF algorithm to make the affects of WoLF more visible. In all cases, both the δ and α learning rates were decreased by a factor inversely proportionate to the iterations through step (2) of the algorithms, although the exact proportion varied between domains and are reported in the appendix. These learning rates were set by hand, but with little or no fine tuning. In practice, an appropriate α learning rate and decay for Q-learning should also work in for WoLF-PHC. The δ learning rates and decay require some amount of care to have them set appropriate for the value of the α learning rate, i.e., we do not want to take larger steps than for which we can update values. Also, all experiments used an *ϵ-projection* exploration with a fixed exploration rate of 5%. This means that only enough uniform randomness is added to the policy at each action selection to insure that all actions are selected with probability $\epsilon/|A_i|$. To satisfy the requirement as a "suitable exploration policy" this should be decayed over time. However, in order to keep things as simple as possible, it was kept constant.

When the results show policy trajectories, these correspond to a single training run of the algorithm showing a prototypical trajectory over time. Many training runs have been performed in each domain, all with very similar trajectories to the ones shown in this work.

4.4.1 Matching Pennies and Rock-Paper-Scissors

The algorithm was applied to the zero-sum matrix games, matching pennies and rock-paper-scissors (see Table 2.1). As a reminder, in both games the Nash equilibrium is a mixed strategy consisting of executing all actions with equal probability. As mentioned, a small learning rate ratio and a large number of trials were used in order to better visualize how the algorithm learns and converges.

Figure 4.3 shows the results of applying both policy hill-climbing and WoLF policy hill-climbing to the matching pennies game. WoLF-PHC quickly begins to oscillate around the equilibrium with ever decreasing amplitude. Without the WoLF modification, PHC oscillates around the equilibrium, but with no appearance of converging. This is even more obvious in the game of rock-paper-scissors. The results in Figure 4.4 show trajectories of the players' strategies in policy space through one million steps. Policy hill-climbing circles the equilibrium policy without any hint of converging, while WoLF policy hill-climbing very nicely spirals toward the equilibrium. The behavior is similar to the theoretically proven behavior of WoLF-IGA.

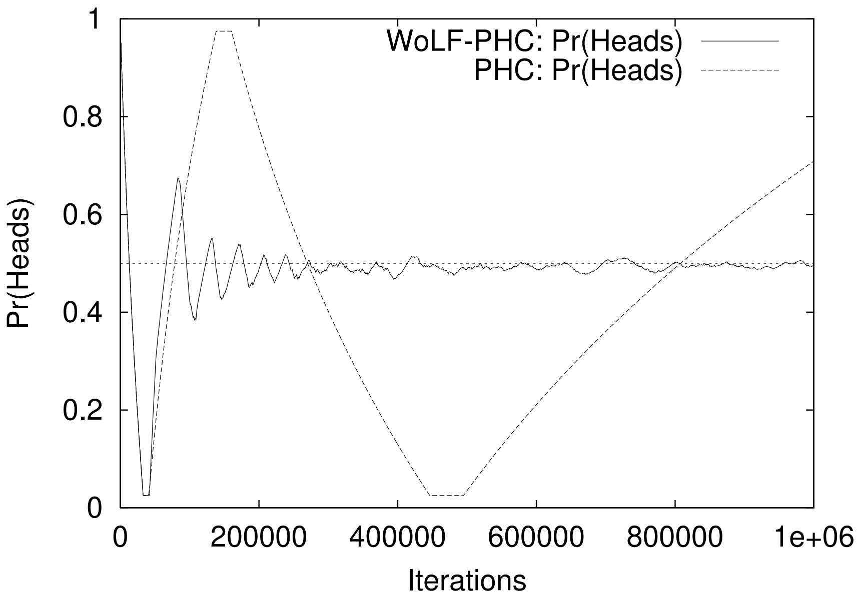


Figure 4.3: Matching pennies: the strategy for one of the players, as a probability distribution, while learning with PHC and WoLF-PHC. The other player's strategy looks similar.

4.4.2 Gridworld

We also examined Hu's grid navigation domain (Hu, 1999), introduced in Section 2.4.1 and reproduced in Figure 4.5. As a reminder, the agents start in two corners and are trying to reach the goal square on the opposite wall. The game is made more interesting by making the North action from the initial state uncertain, only succeeding with probability 0.5. This results in the game having two deterministic equilibria where one player moves laterally on the first move while the other moves North.

WoLF policy hill-climbing successfully converges to one of these equilibria. Figure 4.5 shows an example trajectory of the players' strategies for the initial state while learning over 100,000 steps. In this example the players converged to the equilibrium where Player 1 moves East and Player 2 moves North from the initial state. This is evidence that WoLF policy hill-climbing can learn an equilibrium even in a general-sum game with multiple equilibria. The results are similar for PHC, showing that WoLF is not needed for (nor does it hinder) convergence in this particular game. This demonstrates that PHC's main convergence difficulties are in competitive games or nearly competitive games (as we will see in three player matching pennies). This is not surprising since, as we mentioned in Section 4.1.2, other best-response learners have been shown to converge to equilibria in cooperative or nearly cooperative games. For example, Q-learning, in fact, has been demonstrated to converge to a pure strategy equilibrium in the grid navigation game (Bowling, 2000).

4.4.3 Soccer

The final domain is the comparatively large zero-sum game, grid soccer, introduced by Littman (1994) to demonstrate Minimax-Q. This domain was discussed in Section 2.4.1. Figure 4.6 reproduces an example initial state of this domain. Unlike the grid navigation domain, the Nash equilibrium for this game requires a mixed policy. In fact any deterministic policy can always be defeated.

Our experimental setup resembles that used by Littman (1994). Each player was trained in self-play for one million steps. After training, the player's policy was fixed and a challenger using Q-learning was trained against the player. This determines the learned policy's worst-case performance and gives an idea of how close the player was to the equilibrium policy, which would perform no worse than losing half its games to its challenger. Unlike Minimax-Q, WoLF-PHC and PHC generally oscillate around and through the target solution and so at some points in time may be close to the equilibrium but a short time later be very far from it. In order to account for this, training was continued for another 250,000 steps and evaluated after every 50,000 steps. The worst performing policy was then considered the value of the policy for that learning run. In addition, the Q-values were initialized optimistically, i.e., $\forall s, a\ Q(s, a) = 1$, as is common for encouraging appropriate exploration (Brafman & Tennenholtz, 2002b; Sutton & Barto, 1998).

Figure 4.6 shows the percentage of games won by the different players when playing their challengers. "WoLF" represents WoLF policy hill-climbing trained against another WoLF policy hill-climber. "PHC(L)" and "PHC(W)" represent policy hill-climbing in self-play with $\delta = \delta_l$ and $\delta = \delta_w$, respectively. "WoLF(2x)" represents WoLF policy hill-climbing trained with twice the training (i.e., two million steps). The performance of the policies were averaged over fifty training runs and the standard deviations are shown as the lines beside the bars. The relative ordering by performance is statistically significant.

WoLF-PHC does very well. Its learned policy is close to the equilibrium and continues to improve with more training. The exact effect of the adjustment can be seen by its out-performance of PHC, using either the larger or smaller learning rate. This shows that the success of WoLF is not simply due to changing learning rates, but rather to changing the learning rate at the appropriate time to encourage convergence. Further training for PHC has statistically little effect on either the performance or variance, even up to three million steps of training. This suggests that the learned policies continue to oscillate wildly through the space of possible best-responses.

The speed of convergence of WoLF-PHC is approximately the same as the reported results for Minimax-Q (Littman, 1994). Specifically, Littman reports that in a similarly constructed experiment Minimax-Q with one million steps of self-play training won 37.5% of the games against its challenger[4]. A direct experimental comparison would be needed to make any stronger statements about the rates of convergence. Nevertheless it is compelling that WoLF-PHC is converging to the equilibrium with a speed at all comparable to a learner designed explicitly to learn this equilibrium.

4.4.4 Three Player Matching Pennies

The previous domains only involve two players. We have also examined WoLF in the multiple player matrix game, three player matching pennies, shown in Table 2.1(f). As a reminder, this game has a single Nash equilibrium corresponding to the players randomizing equally between both actions.

Figure 4.7 shows the results of all three players using WoLF-PHC in this domain. The results are shown for two different learning rate ratios: $\delta_l/\delta_w = 2$ and $\delta_l/\delta_w = 3$. Notice that with the larger ratio WoLF converges to the equilibrium just as in the other experiments, but with a lower ratio it does not converge. This is the first domain we have examined where the ratio of the two learning rates seems to be important to WoLF. This is due to the fact that it takes extra time for one player's strategy change to propagate through the extra player before affecting the original player's optimal strategy. It is still very interesting that WoLF converges at all since another sophisticated technique, smooth fictitious play, fails to converge in this game (Fudenberg & Levine, 1999).

The fact that the ratio of learning rates could be important for convergence is not, in theory, a problem. This fact only adds an additional requirement with regards to the decay of the learning rates. In particular, the learning rates should decay in such a way that their ratio approaches infinity, i.e.,

$$\lim_{t\to\infty} \frac{\delta_l(t)}{\delta_w(t)} = \infty.$$

This is in addition to the usual requirements for a gradient following step-size, i.e.,

$$\forall x \in \{l, w\} \qquad \lim_{t\to\infty} \delta_x(t) = 0 \text{ and} \sum_{t=1}^{\infty} \delta_x(t) = \infty.$$

All of these properties can be simultaneously achieved. One example schedule of decay satisfying these properties is,

$$\delta_l(t) = \frac{1}{t^{0.9}} \qquad \delta_w(t) = \frac{1}{t}.$$

Although the dependence of convergence on the ratio of learning rates is not a problem, it is still an interesting and open problem of what games require larger ratios and why.

[4]The results are not directly comparable due to the use of a different decay of the learning rate. Littman's experiments with Minimax-Q used an exponential decay which decreases too quickly for use with WoLF-PHC. Note that with infinite training Minimax-Q provably converges to the equilibrium and win half its games versus its challenger. Our results show that WoLF also seems to be moving closer to the equilibrium with more training.

4.4.5 Results Beyond Self-Play

The previous experiments have been examining WoLF-PHC in self-play, i.e., with all the players using the same algorithm. Corollary 2 (Section 4.2.4) gives a sliver of theoretical evidence that WoLF may learn well against other learners. In this section we present some results examining this empirically.

Matrix Game

The first experiment examining WoLF in situations outside of self-play is the situation described in Corollary 2. We examine the learning of WoLF-PHC against unmodified PHC in rock-paper-scissors. Figure 4.8 shows the trajectories of the players. The corollary predicted convergence, and the proof gave the intuition that the convergence rate would depend on the ratio of the learning rates of both players. In this experiment, we see convergence to the Nash equilibrium, and the convergence is slower than with two WoLF learners (see Figure 4.4(b)) as is consistent with the theoretical results..

Soccer

Since Littman's soccer game (see Figure 4.6) is the most complex game we have examined thus far, we now consider whether WoLF might converge in this domain against different opponents. The first opponent is unmodified policy hill-climbing as above, and the second opponent is Q-Learning. Neither opponents are convergent in self-play, but both are rational. So, if play is going to converge, it must be to a Nash equilibrium. The experimental setup is exactly as in Section 4.4.3. Training is performed for one million steps, and then the resulting policy is frozen and a challenger is trained to find that policy's worst case performance. As before, the closer the policy is to winning half of its games with its challenger, the closer the policy is to the equilibrium.

Figure 4.9 shows the results. As before we show the percentage of games won against its challenger as a measure of the distance from the equilibrium policy. The results are shown for both one million steps of training and twice that amount, both when training against unmodified PHC and against Q-learning. There are two important observations. The first is that the learned policy is comparatively close to the equilibrium (the result of PHC in self-play is also shown for comparison). Second, it appears that more training does move the policy closer to the equilibrium giving evidence of convergence.

None of these empirical results *proves* convergence either in situations of self-play or otherwise. The results do give evidence that a variable learning rate and the WoLF principle can encourage convergence in an otherwise non-convergent rational learning algorithm. Specifically, WoLF-PHC has been shown in a variety of domains to effectively converge to best-response policies, despite the wild non-stationarity of other learning agents using the same or different learning algorithms.

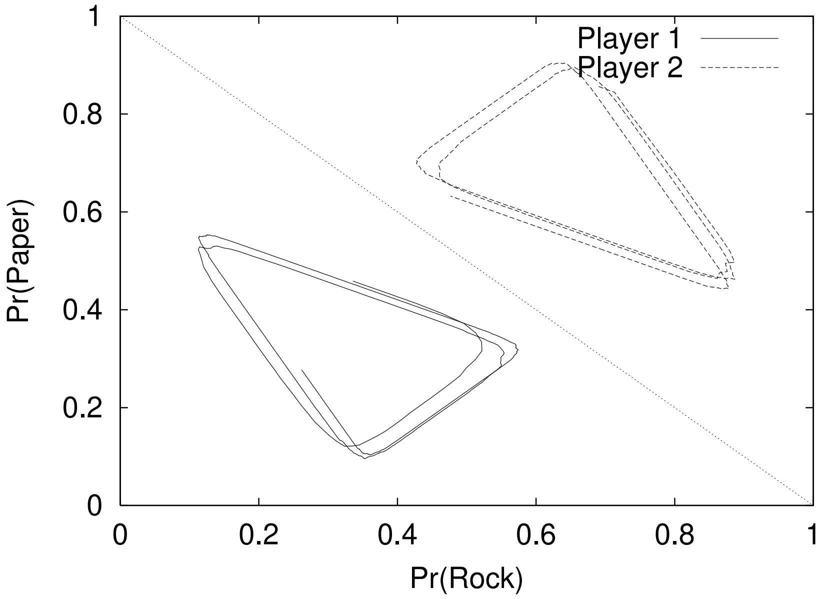
4.5 Summary

In this chapter, we introduced two properties that are desirable for a multiagent learning algorithm: rationality and convergence. We noted that previous algorithms do not simultaneously achieve these properties. We proposed the concept of a variable learning rate toward achieving these properties. In particular we introduced the WoLF, "Win or Learn Fast", principle as a particular variable learning rate.

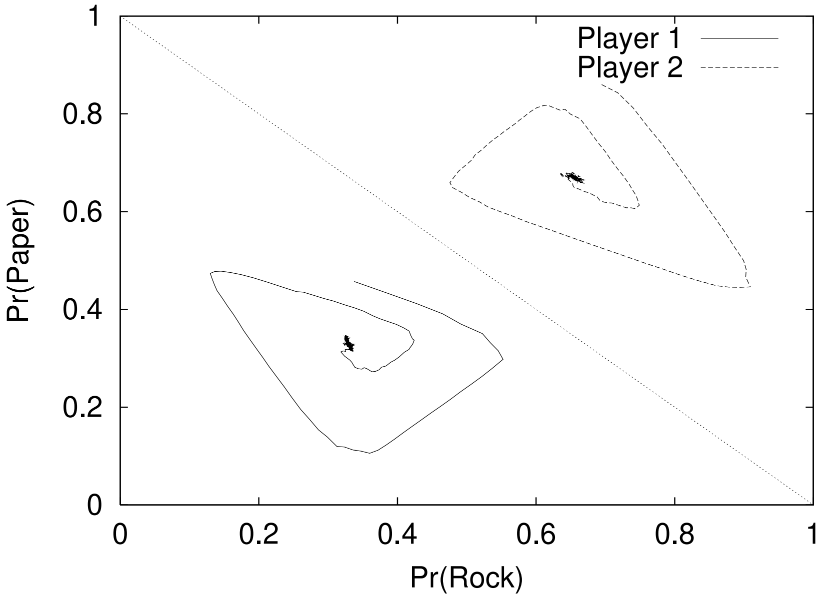
We examined this technique theoretically for learning in two-player, two-action matrix games. We proved that this technique causes infinitesimal gradient ascent, which is not convergent, to converge in these games. We also generalized this result beyond self-play, demonstrating that both players do not need to be using WoLF, in order for it to have a converging effect.

We then describe an implementation of the WoLF principle in a general stochastic game learning algorithm. We then explored this algorithm empirically in a large variety of stochastic games, including both single state and multiple state, zero-sum and non-zero-sum, two-player and three-player games. We also empirically examined several situations beyond self-play including play against a non-WoLF policy hill-climber as well as play against a simple Q-learning algorithm. These results validate the theoretical results that WoLF can cause rational algorithms to converge.

This analysis focused on simple games with few states and actions, where approximation and therefore agent limitations were unnecessary. In the next chapter, we begin our investigation of agent limitations and their effect on games. In Chapter 6 we return to the WoLF principle in the context of learning in the presence of limitations.



(a) Policy Hill-Climbing



(b) WoLF Policy Hill-Climbing

Figure 4.4: Rock-paper-scissors: trajectories of the two players' strategies while learning with (a) PHC, and (b) WoLF-PHC through one million iterations.

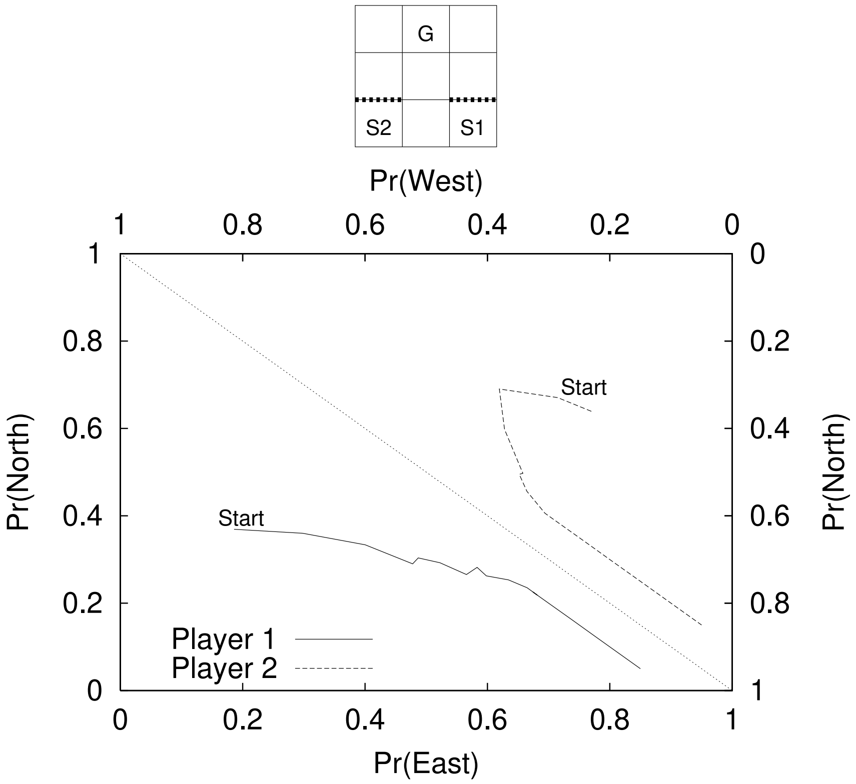


Figure 4.5: Gridworld: trajectories of two players' policies for the initial state while learning with WoLF-PHC. The dashed walls in the grid represent the actions that are uncertain.

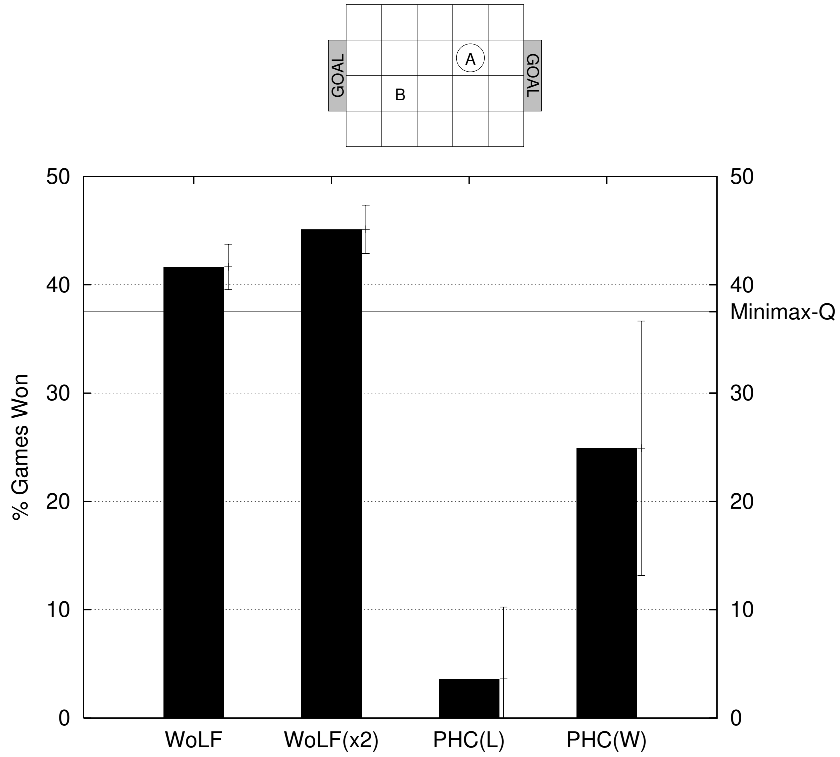
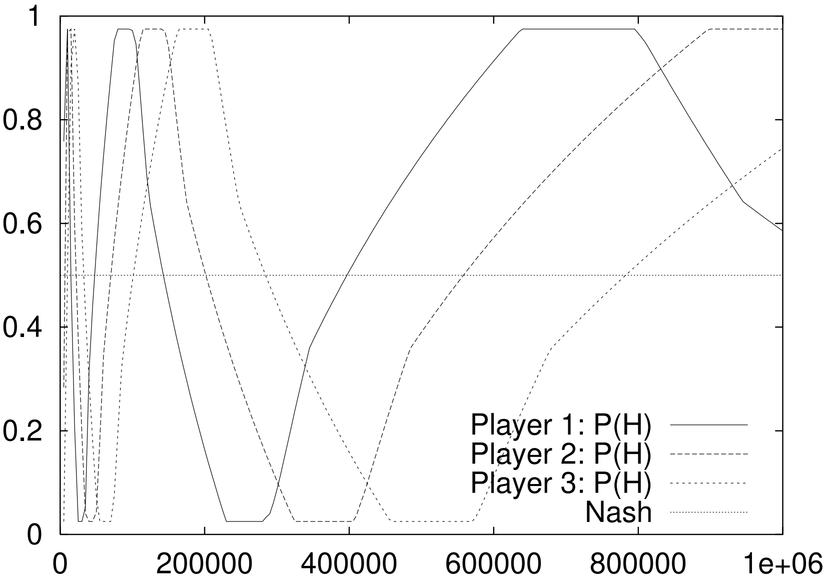
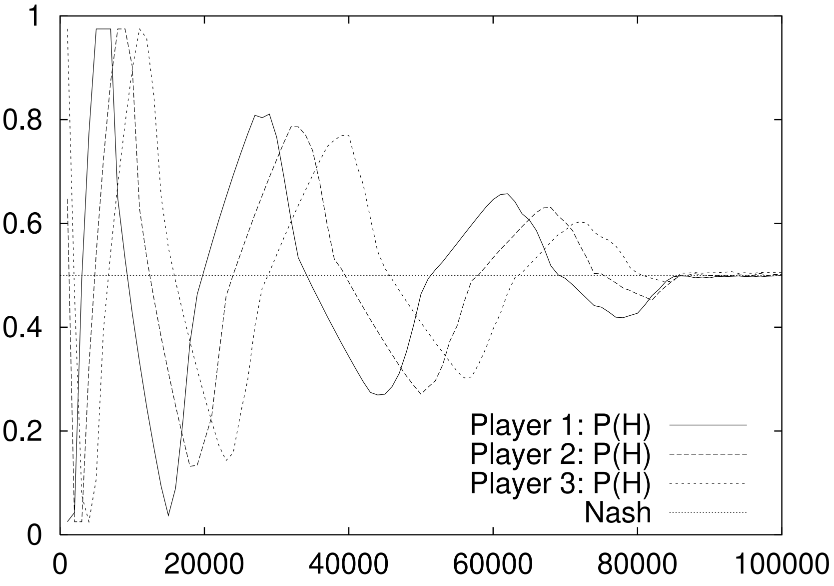


Figure 4.6: Grid soccer: the percentage of games won against a specifically trained worst-case opponent after one million steps of training. The closer this percentage is to 50% the closer the learned policy is to the equilibrium. Error bars are shown and the relative ordering by performance is statistically significant. The reported performance of Minimax-Q (Littman, 1994) is shown by the solid line.



(a) $\delta_l/\delta_w = 2$



(b) $\delta_l/\delta_w = 3$

Figure 4.7: Three player matching pennies: trajectories of strategies of the three players using WoLF-PHC with different ratios of learning rates. Each line corresponds to one player's strategy over time. Notice that the axes have different scales.

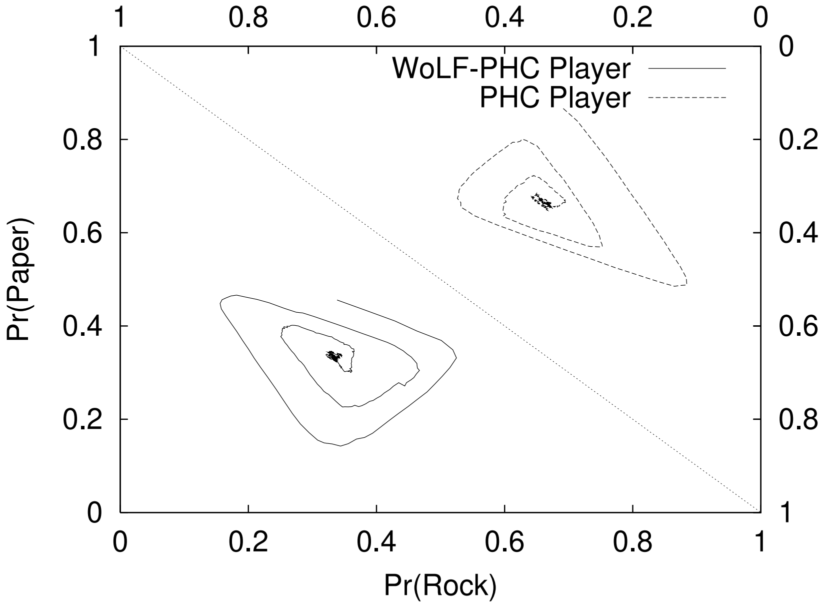


Figure 4.8: Rock-paper-scissors: the trajectories of the players' strategies, one using WoLF-PHC, and the other using PHC, in a prototypical run with one million iterations.

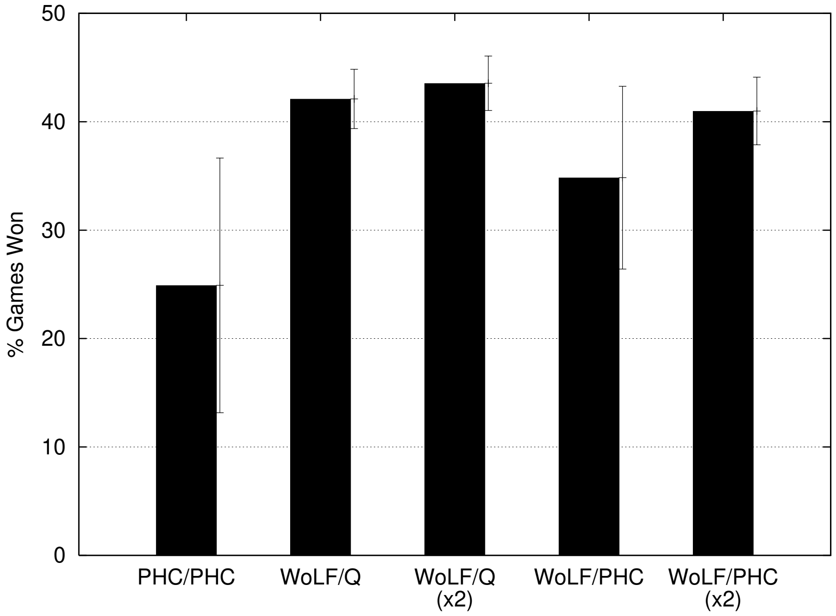


Figure 4.9: Grid soccer: the percentage of games won against a specifically trained worst-case opponent after one million steps of training with a non-identical opponent. The closer this percentage is to 50% the closer the learned policy is to the equilibrium. Notice the added improvement with more training. PHC in self-play is included for comparison purposes.

Chapter 5

Analyzing Limitations

The solution concept of Nash equilibria depends on all of the agents playing optimally. From the agent development perspective, agents have limitations that prevent this from being a reality. In the previous chapter we examined abstract stochastic games with a small number of states. In these domains a consideration of limitations was unnecessary. In order for learning algorithms to be applied to more interesting and realistic problems, a consideration of limitations is required.

This analysis is not only useful for our further examination of learning with limitations in Chapter 6, but is an important consideration for multiagent learning, in general. The solution concept of Nash equilibria is a critical foundation for most of the multiagent learning research to date. Equilibrium learners obviously rely critically on the existence of equilibria in order to have an explicit solution to learn[1]. Best-response learners also depend on the existence of equilibria, since these are the only possible convergence points when playing other best-response learning agents. Therefore, this analysis is critical to any exploration of multiagent learning in realistic problems, where agents are necessarily limited.

In this chapter we analyze the affect of agent limitations on the Nash equilibrium concept in stochastic games. We begin by examining the source of common agent limitations. Following this enumeration, we propose two different models for the effect of limitations on agent behavior: *implicit games* and *restricted policy spaces*. We then use these models to explore the impact limitations have on the Nash equilibrium solution concept. We define the notion of *restricted equilibria*, which explicitly accounts for agent limitations. We then analyze theoretically under what conditions restricted equilibria are guaranteed to exist. Finally, we explore learning in simple cases where restricted equilibria are guaranteed to exist. We show empirically, that the WoLF-PHC algorithm from Section 4.3.2 is capable of learning restricted equilibria when they exist.

[1]It should be noted that in the case of Minimax-Q, the algorithm and solution concept are still well-defined. A policy that maximizes its worst-case value may still exist even if limitations make it such that no equilibrium exists. However, this minimax optimal policy might not be part of any equilibrium. Later (in Section 5.2, Fact 5), we present an example of a zero-sum stochastic game and agent limitations where the minimax optimal policies exist but do not comprise an equilibrium.

5.1 Limitations

The working definition of limitation in this work is *anything that can restrict the agent from learning or playing optimal policies*. Broadly speaking, limitations can be classified into two categories: physical limitations and rational limitations. Physical limitations are those caused by the interaction of the agent with its environment and are often unavoidable. Rational limitations are limitations specifically chosen by the agent designer to make the learning problem tractable, either in memory or time. We briefly explore some of these limitations informally before presenting a formal model of limitations that attempts to capture their effect within the stochastic game framework.

5.1.1 Physical Limitations

One obvious physical limitation is that the agent simply is broken. A mobile agent may cease to move or less drastically may lose the use of one of its actuators preventing certain movements. Similarly, another agent may appear to be "broken" when in fact the motion is simply outside its capabilities. For example, in a mobile robot environment where the "rules" allow robots to move up to two meters per second, there may be a robot that is not capable of reaching that speed. An agent that is not broken, may suffer from poor control where its actions are not always carried out as desired, e.g., due to poorly tuned servos, inadequate wheel traction, or high system latency.

Another common physical limitation is hardwired behavior. Most agents in dynamic domains need some amount of hard-wiring for fast response and safety. For example, many mobile robot platforms are programmed to immediately stop if an obstacle is too close. These hardwired actions prevent certain behavior by the agent, which is often unsafe but is potentially optimal.

Sensing is a common area of agent limitations encompassing everything from noise to partial observability. Here we'll mention just one broad category of sensing problems: state aliasing. This occurs when an agent cannot distinguish between two different states of the world. An agent may need to remember past states and actions in order to properly distinguish the states, or may simply execute the same action in both states.

5.1.2 Rational Limitations

Rational limitations are a requirement for agents to learn in even moderately sized problems. Techniques for making learning scale, which often focus on near-optimal solutions, continue to be proposed and investigated in the context of single-agent learning. These techniques for scaling are likely to be more necessary in multiagent environments which tend to have larger state spaces. We now examine a few specific methods.

In domains with sparse rewards one common technique is reward shaping, e.g., (Mataric, 1994). A designer artificially rewards the agent for actions the designer believes to be progressing toward the sparse rewards. This can often speed learning by focusing exploration, but also can cause the agent to learn suboptimal policies. For example, in robotic soccer moving the ball down the field is a good heuristic for goal progression, but at times the optimal goal-scoring policy is to pass the ball backwards to an open teammate.

Subproblem reuse also has a similar effect, where a subgoal is used in a portion of the state space to speed learning, e.g., (Hauskrecht, Meuleau, Kaelbling, Dean, & Boutilier, 1998; Bowling & Veloso, 1999). These subgoals, though, may not be optimal for the global problem and so prevent the agent from playing optimally. Temporally abstract options, either provided (Sutton, Precup, & Singh, 1998) or learned (McGovern & Barto, 2001; Uther, 2002), also enforce a particular sub-policy on a portion of the state space. Although in theory, the primitive actions are still available to the agents to play optimal policies, in practice, abstracting away primitive actions is often necessary in large or continuous state spaces.

Parameterized policies are receiving a great deal of attention as a way for reinforcement learning to scale to large problems, e.g., (Williams & Baird, 1993; Sutton et al., 2000; Baxter & Bartlett, 2000). The idea is to give the learner a policy that depends on far less parameters than the entire policy space actually would require. Learning is then performed in this smaller space of parameters using gradient techniques. This simplifies and speeds learning at the expense of possibly not being able to represent the optimal policy in the parameter space.

5.1.3 Models of Limitations

This enumeration of limitations shows that there are a number and variety of limitations with which agents may be faced, and they cannot be realistically avoided. In order to understand their impact on equilibria we model limitations formally within the stochastic game framework. We introduce two models that capture broad classes of limitations: *implicit games* and *restricted policy spaces*.

Implicit Games. Limitations may cause an agent to play suboptimally, but it may be that the agent *is* actually playing optimally in a different game. If this new game can be defined within the stochastic game framework we call this an *implicit game*, in contrast to the original game called the *explicit game*. For example, reward shaping adds artificial rewards to help guide the agent's search. Although the agent is no longer learning an optimal policy in the explicit game, it is learning an optimal policy of some game, specifically the game with these additional rewards added to that agent's R_i function. Another example is due to broken actuators preventing an agent from taking some action. The agent may be suboptimal in the explicit game, while still being optimal in the implicit game defined by removing these actions from the agent's action set, A_i. We can formalize this concept in the following definition.

Definition 8 *Given a stochastic game* $(n, \mathcal{S}, \mathcal{A}_{1\ldots n}, T, R_{1\ldots n})$ *the tuple* $(n, \mathcal{S}, \hat{\mathcal{A}}_{1\ldots n}, \hat{T}, \hat{R}_{1\ldots n})$ *is an implicit game if and only if it is itself a stochastic game and there exist mappings,*

$$\tau_i : \mathcal{S} \times \hat{\mathcal{A}}_i \times \mathcal{A}_i \rightarrow [0, 1],$$

such that,

$$\forall s, s' \in \mathcal{S}\ \forall \hat{a}_i \in \hat{\mathcal{A}}_i \qquad \hat{T}(s, \langle \hat{a}_i \rangle_{i=1\ldots n}, s') = \sum_{a \in \mathcal{A}} \Pi_{i=1}^{n} \tau_i(s, \hat{a}_i, a_i)\, T(s, \langle a_i \rangle_{i=1\ldots n}, s').$$

The τ_i*'s are mappings from the implicit action space into stochastic actions in the explicit action space.*

Reward shaping, broken actuators, and exploration can all be captured within this model. For reward shaping the implicit game is $(n, \mathcal{S}, \mathcal{A}_{1\ldots n}, T, \hat{R}_{1\ldots n})$, where $\hat{R}_i$ adds the shaped reward into the original reward, R_i. In this case the τ mappings are just the identity, $\tau_i(s,a) = a$. For the broken actuator example, let $a_i^0 \in \mathcal{A}_i$ be some null action for agent i and let $a_i^b \in \mathcal{A}_i$ be some broken action for agent i that under the limitation has the same effect as the null action. The implicit game, then, is $(n, \mathcal{S}, \mathcal{A}_{1\ldots n}, \hat{T}, \hat{R}_{1\ldots n})$, where,

$$\begin{aligned}
\hat{T}(s,a,s') &= \begin{cases} T(s, \langle a_i^0, a_{-i}\rangle, s') & \text{if } a_i = a_i^b \\ T(s,a,s') & \text{otherwise} \end{cases} \\
\hat{R}(s,a) &= \begin{cases} R(s, \langle a_i^0, a_{-i}\rangle) & \text{if } a_i = a_i^b \\ R(s,a) & \text{otherwise} \end{cases},
\end{aligned}$$

and,

$$\tau_i(s,a) = \begin{cases} a_i^0 & \text{if } a = a_i^b \\ a & \text{otherwise} \end{cases}.$$

For ϵ exploration, as with broken actuators, only $\hat{T}$ and $\hat{R}$ need to be defined. They become just the ϵ combination of T or R with the transition probabilities or reward values of selecting a random action.

Limitations captured by this model can easily be analyzed with respect to their effect on the existence of equilibria. We now restrict ourselves to discounted reward stochastic games, where equilibria existence is known. Using the intuitive definition of an equilibrium as a joint policy such that "no player can do better by changing policies," an equilibrium in the implicit game achieves this definition for the explicit game. Since all discounted reward stochastic games have at least one equilibrium, so must the implicit game, which is also in this class. This equilibrium for the implicit game is then an equilibrium in the explicit game, given that the agents are limited.

On the other hand, many of the limitations described above cannot be modeled in this way. None of the limitations of abstraction, subproblem reuse, parameterized policies, or state aliasing lend themselves to being described by this model. This leads us to our second, and in many ways more general, model of limitations.

Restricted Policy Spaces. The second model is that of *restricted policy spaces*, which models limitations as restricting the agent from playing certain policies. For example, a fixed exploration strategy restricts the player to policies that select all actions with some minimum probability. Parameterized policy spaces have a restricted policy space corresponding to the space of policies that can be represented by their parameters. We can define this formally.

Definition 9 *A* restricted policy space *for player i is a non-empty and compact*[2] *subset, $\overline{\Pi}_i \subseteq \Pi_i$.*

It should be straightforward to see that parameterized policies, exploration, state aliasing (with no memory), and subproblem reuse all can be captured as a restriction on policies that the agent can play. Therefore they can be naturally described as restricted policy spaces. In addition, value

[2]Since $\overline{\Pi}_i$ is a subset of a bounded set, the requirement that $\overline{\Pi}_i$ is compact merely adds that the limit point of any sequence of elements from the set is also in the set. The assumption of compactness may at first appear strange, but it is not particularly limiting, and is critical for any equilibrium analysis.

Physical Limitations	**Implicit Games**	**Restricted Policies**
Broken Actuators	✓	✓
Hardwired Behavior	✓	✓
Poor Control		✓
State Aliasing		✓
Rational Limitations	**Implicit Games**	**Restricted Policies**
Reward Shaping or Incentives	✓	
Exploration	✓	✓
State Abstraction/Options		✓
Subproblems		✓
Parameterized Policy		✓

Table 5.1: Common agent limitations. The column check-marks correspond to whether the limitation can be modeled straightforwardly using implicit games and/or restricted policy spaces.

function approximation can also be modeled as a restricted policy space. Value function reinforcement learning involves some mapping from values to a distribution over actions, such as ϵ-greedy or Boltzmann mappings. Since value function approximation limits the space of values, it correspondingly limits the space of policies.

Unlike implicit games, an analysis of the existence of equilibria under the model of restricted policy spaces is not at all straightforward. Since the model captures most of the really interesting limitations, such an analysis is precisely our focus of the next section. Before moving on to this investigation, though, we summarize our enumeration of limitations in Table 5.1. The limitations that we have discussed are listed, including which model most naturally captures their effect on agent behavior.

5.2 Restricted Equilibria

In this section we define formally the concept of *restricted equilibria*, which account for agents' restricted policy spaces. We then carefully analyze what can be proven about the existence of restricted equilibria. The results presented range from somewhat trivial examples (Facts 1, 2, 3, and 4) and applications of known results from game theory and basic analysis (Theorems 3 and 7) to results that we believe are completely new (Theorems 4, 5, and 6), as well as a critical counterexample to the wider existence of restricted equilibria (Fact 5). But all of the results are in a sense novel since this specific question has received no direct attention in the game theory nor the multiagent learning literature.

5.2.1 Definition

We begin by defining the concept of equilibria under the model of restricted policy spaces. First, we need a notion of best-response that accounts for the players' limitations.

Definition 10 *A* restricted best-response *for player i, $\overline{\mathrm{BR}}_i(\pi_{-i})$, is the set of all policies from $\overline{\Pi}_i$*

that are optimal given the other player(s) play the joint policy π_{-i}.

In our consideration of limitations, it is necessary to use a more relaxed definition of optimality for the discounted reward framework than what was presented in Chapter 2. In particular, it may be the case with limitations that it is not possible to simultaneously maximize the value at every state. That is, for some policy π_{-i}, there may not exist any policy, $\pi_i \in \overline{\Pi}_i$, such that,

$$\forall \pi'_i \in \overline{\Pi}_i \; \forall s \in \mathcal{S} \qquad V_i^{\langle \pi_i, \pi_{-i} \rangle}(s) \geq V_i^{\left\langle \pi'_i, \pi_{-i} \right\rangle}(s).$$

Instead of this stronger notion, we relax optimality to only require an optimal policy to maximize the value of some specified initial state, $s_0 \in \mathcal{S}$. We can define this formally.

Definition 11 *Throughout this chapter unless otherwise stated, we say that* $pi_i \in \overline{\Pi}_i$ *is optimal given the other player(s) play the joint policy* π_{-i}*, if and only if,*

$$\forall \pi'_i \in \overline{\Pi}_i \qquad V_i^{\langle \pi_i, \pi_{-i} \rangle} \geq V_i^{\left\langle \pi'_i, \pi_{-i} \right\rangle},$$

where $V_i^{\pi} = V_i^{\pi}(s_0)$ *under either the average reward or discounted reward frameworks.*

We now use our notion of a best-response set to define an equilibrium.

Definition 12 *A* restricted equilibrium *is a joint policy,* $\pi_{i=1\ldots n}$*, where,*

$$\pi_i \in \overline{\mathrm{BR}}_i(\pi_{-i}).$$

So, no player can within their restricted policy space do better by changing policies given that all the other players continue to follow the equilibrium policy.

5.2.2 Existence of Restricted Equilibria

We can now state some results about when the existence of equilibria are preserved by restricted policy spaces, and when they are not. Unless otherwise stated (as in Theorems 4 and 6, which only apply to discounted reward), the results presented here apply equally to both the discounted reward and the average reward formulations. We will separate the proofs for the two reward formulations when needed. The first four facts show that the question of the existence of restricted equilibria does not have a trivial answer.

Fact 1 *Restricted equilibria do not necessarily exist.*

Proof. Consider the rock-paper-scissors matrix game with players restricted to the space of deterministic policies. There are nine joint deterministic policies, and none of these joint policies are a restricted equilibrium. □

Fact 2 *There exist restricted policy spaces such that restricted equilibria exist.*

Proof. One trivial restricted equilibrium is in the case where all agents have a singleton policy subspace. The singleton joint policy must be a restricted equilibrium. □

Fact 3 *If π^* is a Nash equilibrium and $\pi^* \in \overline{\Pi}$, then π^* is a restricted equilibrium.*

Proof. If π^* is a Nash equilibrium, then we have

$$\forall i \in \{1 \ldots n\} \; \forall \pi_i \in \Pi_i \qquad V_i^{\pi^*} \geq V_i^{\langle \pi_i, \pi^*_{-i} \rangle}.$$

Since $\overline{\Pi}_i \subseteq \Pi_i$, then we also have

$$\forall i \in \{1 \ldots n\} \; \forall \pi_i \in \overline{\Pi}_i \qquad V_i^{\pi^*} \geq V_i^{\langle \pi_i, \pi^*_{-i} \rangle},$$

and thus π^* is a restricted equilibrium. □

On the other hand, the converse is not true; not all restricted equilibria are of this trivial variety.

Fact 4 *There exist non-trivial restricted equilibria that are neither Nash equilibria nor come from singleton policy spaces.*

Proof. Consider the rock-paper-scissors matrix game from Table 2.1(b). Suppose the column player is forced, due to some limitation, to play "Paper" exactly half of the time, but is free to choose between "Rock" and "Scissors" otherwise[3]. This limitation is a restricted policy space that excludes the only Nash equilibrium of the game. We can solve this game using the implicit game model, by giving the limited player only two actions, $s_1 = (0.5, 0.5, 0)$ and $s_2 = (0, 0.5, 0.5)$, which the player can mix between. This is depicted graphically in Figure 5.1. We can solve the implicit game and convert the two actions back to actions of the explicit game to find a restricted equilibrium. Notice this restricted equilibrium is not a Nash equilibrium. □

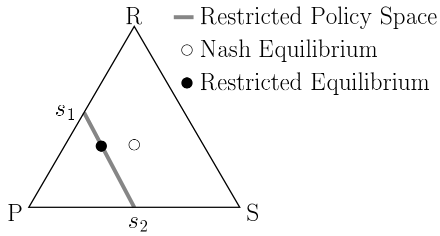
The Fact 4 example has a convex policy space, i.e., all linear combinations of policies in the set are also in the set. The Fact 1 counterexample has a non-convex policy space. This suggests that restricted equilibria may exist as long as the restricted policy space is convex. We can prove this for repeated games, but unfortunately it is not generally true for stochastic games.

Theorem 3 *When $|S| = 1$, i.e., in repeated games, if $\overline{\Pi}_i$ is convex, then there exists a restricted equilibrium.*

Proof. One might think of proving this by appealing to implicit games as was used in Fact 4. If $\overline{\Pi}_i$ was a convex hull of a *finite* number of strategies, then this would be possible. In order to prove it for any convex $\overline{\Pi}_i$, we apply Rosen's theorem about the existence of equilibria in concave games (Rosen, 1965). In order to use this theorem we need to show the following:

1. $\overline{\Pi}_i$ is non-empty, compact, and convex.
2. V_i^{π} as a function over $\pi \in \overline{\Pi}$ is continuous.

[3]This is an example of reducing the number of parameters that need to be learned, in this case from two to one. Although this reduction of parameters is small, it illustrates the use of limitations to simplify and, therefore, speed learning.



	Explicit Game	Implicit Game
Payoffs	$\begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} -\frac{1}{2} & 0 \\ \frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix}$
Nash Equilibrium	$\left\langle \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right\rangle, \left\langle \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right\rangle$	$\left\langle 0, \frac{1}{3}, \frac{2}{3} \right\rangle, \left\langle \frac{2}{3}, \frac{1}{3} \right\rangle$
Restricted Equilibrium	$\left\langle 0, \frac{1}{3}, \frac{2}{3} \right\rangle, \left\langle \frac{1}{3}, \frac{1}{2}, \frac{1}{6} \right\rangle$	

Figure 5.1: Example of a restricted equilibrium that is not a Nash equilibrium. Here, the column player in rock-paper-scissors is restricted to playing only linear combinations of the strategies $s_1 = \left\langle \frac{1}{2}, \frac{1}{2}, 0 \right\rangle$ and $s_2 = \left\langle 0, \frac{1}{2}, \frac{1}{2} \right\rangle$.

3. For any $\pi \in \overline{\Pi}$, the function over $\pi'_i \in \overline{\Pi}_i$ defined as $V_i^{\langle \pi'_i, \pi_{-i} \rangle}$ is concave.

Condition 1 is by assumption. In repeated games, where $\mathcal{S} = \{s_0\}$, we can simplify the definition of a policy's value from Equations 2.11 and 2.12. For discounted reward we get,

$$V_i^\pi \quad = \quad \frac{1}{1-\gamma} \sum_{a \in \mathcal{A}} R_i(s, a) \Pi_{i=1}^n \pi_i(s_0, a_i), \tag{5.1}$$

For average reward, the value is just the value of the one-shot matrix game given the players' policies. This is equivalent to setting $\gamma = 0$ in Equation 5.1. Equation 5.1 shows that the value is a multilinear function with respect to the joint policy and therefore is continuous. So Condition 2 is satisfied. Observe that by fixing the policies for all but one player Equation 5.1 becomes a linear function over the remaining player's policy and so is also concave satisfying Condition 3. Therefore Rosen's theorem applies and this game has a restricted equilibrium. □

Fact 5 *For a stochastic game, even if $\overline{\Pi}_i$ is convex, restricted equilibria do not necessarily exist.*

Proof. Consider the stochastic game in Figure 5.2. This is a zero-sum game where only the payoffs to the row player are shown. The discount factor is $\gamma \in (0, 1)$. The actions available to the row player are U and D, and for the column player L and R. From the initial state, s_0, the column player may select either L or R which results in no rewards but with high probability, $1 - \epsilon$, transitions to the specified state (regardless of the row player's action), and with low probability, ϵ, transitions to the opposite state. In each of the resulting states the players play the matrix game shown and then

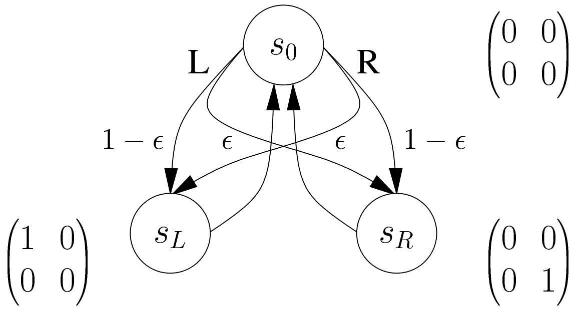


Figure 5.2: An example stochastic game where convex restricted policy spaces do not preserve the existence of equilibria.

deterministically transition back to the initial state. Notice that this game is unichain, where all the states are in a single ergodic set, thus satisfying the average reward formulation requirement.

Now consider the restricted policy space where players have to play their actions with the same probability in all states. So,

$$\overline{\Pi}_i \quad = \quad \{\pi_i \in \Pi_i | \forall s, s' \in \mathcal{S}\ \forall a \in \mathcal{A} \quad \pi_i(s,a) = \pi_i(s',a)\}\,. \tag{5.2}$$

Notice that this is a convex set of policies. That is, if policies x_1 and x_2 are in $\overline{\Pi}_i$ (according to Equation 5.2), then for any $\alpha \in [0,1]$, x_3 must also be in $\overline{\Pi}_i$, where,

$$x_3(s,a) = \alpha x_1(s,a) + (1-\alpha)x_2(s,a). \tag{5.3}$$

This can be seen by examining $x_3(s',a)$ for any $s' \in \mathcal{S}$. From Equation 5.3, we have,

$$\begin{aligned} x_3(s',a) &= \alpha x_1(s',a) + (1-\alpha)x_2(s',a) && (5.4)\\ &= \alpha x_1(s,a) + (1-\alpha)x_2(s,a) && (5.5)\\ &= x_3(s,a). && (5.6)\end{aligned}$$

Therefore, x_3 is in $\overline{\Pi}_i$ and hence $\overline{\Pi}_i$ is convex.

This game, though, does not have a restricted equilibrium. The four possible joint deterministic policies, (U,L), (U,R), (D,L), and (D,R), are not equilibria. So if there exists an equilibrium it must be mixed. Consider any mixed strategy for the row player. If this strategy plays U with probability less than $\frac{1}{2}$ then the unique best-response for the column player is to play L; if greater than $\frac{1}{2}$ then the unique best-response is to play R; if equal *then the unique best-responses are to play L or R deterministically*. In all cases all best-responses are deterministic, so this rules out mixed strategy equilibria, and so no equilibria exists. □

Convexity is not a strong enough property to guarantee the existence of restricted equilibria. Standard equilibrium proof techniques fail for this example due to the fact that the player's best-response sets are not convex, even though their restricted policy spaces are convex. Notice that the best-response to the row player mixing equally between actions is to play either of its actions deterministically, but, linear combinations of these actions (e.g., mixing equally) are not best-responses.

This intuition is proven in the following lemma.

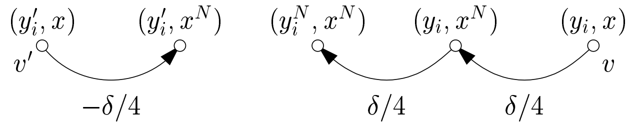


Figure 5.3: An illustration of the demonstration by contradiction that the best-response functions are upper hemi-continuous.

Lemma 10 *For any stochastic game, if $\overline{\Pi}_i$ is convex and for all $\pi_{-i} \in \overline{\Pi}_{-i}$, $\overline{\mathrm{BR}}_i(\pi_{-i})$ is convex, then there exists a restricted equilibrium.*

Proof. The proof relies on Kakutani's fixed point theorem. We first need to show some facts about the restricted best-response function. First, remember that $\overline{\Pi}_i$ is non-empty and compact. Also, note that the value (with both discounted and average reward) to a player at any state of a joint policy is a continuous function of that joint policy (Filar & Vrieze, 1997, Theorem 4.3.7 and Lemma 5.1.4). Therefore, from basic analysis (Gaughan, 1993, Theorem 3.5 and Corollary 3.11), the set of maximizing (or optimal) points must be a non-empty and compact set. So $\overline{\mathrm{BR}}_i(\pi_{-i})$ is non-empty and compact.

Define the set-valued function, $F : \overline{\Pi} \to \overline{\Pi}$,

$$F(\pi) = \times_{i=1}^{n} \overline{\mathrm{BR}}_i(\pi_{-i}).$$

We want to show F has a fixed point. To apply Kakutani's fixed point theorem we must show the following conditions to be true,

1. $\overline{\Pi}$ is a non-empty, compact, and convex subset of a Euclidean space,
2. $F(\pi)$ is non-empty,
3. $F(\pi)$ is compact and convex, and
4. F is upper hemi-continuous.

Since the Cartesian product of non-empty, compact, and convex sets is non-empty, compact, and convex we have condition (1) by the assumptions on $\overline{\Pi}_i$. By the facts of $\overline{\mathrm{BR}}_i$ from above and the lemma's assumptions we similarly get conditions (2) and (3).

What remains is to show condition (4). Consider two sequences $x^j \to x \in \overline{\Pi}$ and $y^j \to y \in \overline{\Pi}$ such that $\forall j\ y^j \in F(x^j)$. It must be shown that $y \in F(x)$, or just $y_i \in \overline{\mathrm{BR}}_i(x)$. Let v be y_i's value against x. By contradiction assume there exists a y_i' with higher value, v' than y_i; let $\delta = v' - v$. Since the value function is continuous we can choose an N large enough that the value of y_i' against x^N differs from v' by at most $\delta/4$[4], and the value of y_i against x^N differs from v by at most $\delta/4$, and the value of y_i^N against x^N differs from y_i against x^N by at most $\delta/4$. The comparison of values of these various joint policies is shown in Figure 5.3. Adding all of these together, we have a point in the sequence $y_i^{n>N}$ whose value against x^n is less than the value of y_i against x^n. So $y_i^n \notin \overline{\mathrm{BR}}_i(x^n)$, and therefore $y^n \notin F(x^n)$ creating our contradiction.

We can now apply Kakutani's fixed point theorem. So there exists $\pi \in \overline{\Pi}$ such that $\pi \in F(\pi)$. This means $\pi_i \in \overline{\mathrm{BR}}_i(\pi_{-i})$, and therefore this is a restricted equilibrium. □

[4]This value is arbitrarily selected and is only required to be strictly smaller than $\delta/3$.

The consequence of this lemma is that, if we can prove that the sets of restricted best-responses are convex then restricted equilibria exist. As we have stated earlier this was not true of the counterexample in Fact 5. The next four theorems all further limit either the restricted policy spaces or the stochastic game to situations where the best-response sets are provably convex. We will first examine a specific class of restricted policy spaces, and then examine specific classes of stochastic games.

A Subclass of Restricted Policies

Our first result for general stochastic games uses a stronger notion of convexity for restricted policy spaces.

Definition 13 *A restricted policy space* $\overline{\Pi}_i$ *is* statewise convex *if and only if it is the Cartesian product over all states of convex strategy sets. Equivalently, if for all* $x_1, x_2 \in \overline{\Pi}_i$ *and all functions* $\alpha : \mathcal{S} \to [0, 1]$*, the policy* $x_3(s, a) = \alpha(s)x_1(s, a) + (1 - \alpha(s))x_2(s, a)$ *is also in* $\overline{\Pi}_i$*.*

Effectively, this amounts to the probability distribution over actions at each state must be specifiable independently of the distribution specified for other states. In this case, we can guarantee that a restricted equilibrium exists.

Theorem 4 *In the discounted reward formulation, if* $\overline{\Pi}_i$ *is statewise convex, then there exists a restricted equilibrium.*

Proof. With statewise convex policy spaces, there exist optimal policies in the strong sense as was originally presented in Section 2.4.2. Specifically, there exists a policy that can simultaneously maximize the value of all states. Formally, for any π_{-i} there exists a $\pi_i \in \overline{\Pi}_i$ such that,

$$\forall s \in \mathcal{S}\ \forall \pi'_i \in \overline{\Pi}_i \qquad V_i^{\langle \pi_i, \pi_{-i} \rangle}(s) \geq V_i^{\left\langle \pi'_i, \pi_{-i} \right\rangle}(s).$$

Suppose this were not true, i.e., there were two policies each which maximized the value of different states. We can construct a new policy that in each state follows the policy whose value is larger for that state. This policy will maximize the value of both states that those policies maximized, and due to statewise convexity is also in $\overline{\Pi}_i$. We will use that fact to redefine optimality to the strong sense for this proof.

We now make use of Lemma 10. First, notice the lemma's proof still holds even with this new definition of optimality. We just showed that under this redefinition, $\overline{\mathrm{BR}}_i(\pi_{-i})$ is non-empty, and the same argument for compactness of $\overline{\mathrm{BR}}_i(\pi_{-i})$ holds. So we can make use of Lemma 10 and what remains is to prove that $\overline{\mathrm{BR}}_i(\pi_{-i})$ is convex. Since π_{-i} is a fixed policy for all of the other players, this defines an MDP for player i (Filar & Vrieze, 1997, Corollary 4.2.11). So we need to show that the set of polices from the player's restricted set that are optimal for this MDP is a convex set. Concretely, if $x_1, x_2 \in \overline{\Pi}$ are optimal for this MDP, then the policy $x_3(s, a) = \alpha x_1(s, a) + (1 - \alpha)x_2(s, a)$ is also optimal for any $\alpha \in [0, 1]$. Since x_1 and x_2 are optimal in the strong sense, i.e., maximizing the value of all states simultaneously, then they must have the same per-state value.

Here, we will use the notation $V^x(s)$ to refer to the value of policy x from state s in this fixed MDP. The value function for any policy satisfies the Bellman equations, specifically,

$$\forall s \in \mathcal{S} \qquad V^x(s) \;=\; \sum_a x(s,a)\left(R(s,a) + \gamma \sum_{s'} T(s,a,s')V^x(s')\right). \tag{5.7}$$

For x_3 then we get the following,

$$\begin{aligned}
V^{x_3}(s) &= \sum_a x_3(s,a)\left(R(s,a) + \gamma \sum_{s'} T(s,a,s')V^{x_3}(s')\right) && (5.8)\\
&= \sum_a (\alpha x_1(s,a) + (1-\alpha)x_2(s,a))\left(R(s,a) + \gamma \sum_{s'} T(s,a,s')V^{x_3}(s')\right) && (5.9)\\
&= \alpha \sum_a x_1(s,a)\left(R(s,a) + \gamma \sum_{s'} T(s,a,s')V^{x_3}(s')\right) + \\
&\quad (1-\alpha)\sum_a x_2(s,a))\left(R(s,a) + \gamma \sum_{s'} T(s,a,s')V^{x_3}(s')\right). && (5.10)
\end{aligned}$$

Notice that $V^{x_3}(s) = V^{x_1}(s) = V^{x_2}(s)$ is a solution to these equations, and therefore is the unique solution for the value of x_3. Therefore, x_3 has the same values as x_1 and x_2, and hence is also optimal. Therefore $\overline{\mathrm{BR}}_i(\pi_{-i})$ is convex, and from Lemma 10 we get the existence of restricted equilibria under this stricter notion of optimality, which also makes the policies a restricted equilibrium under our original notion of optimality, that is, only maximizing the value of the initial state. □

Subclasses of Stochastic Games

Unfortunately, most rational limitations that allow reinforcement learning to scale are not statewise convex restrictions, and usually have some dependence between states. For example, parameterized policies involve far fewer parameters than the number of states, which can be intractably large, and so the space of policies cannot select actions at each state independently. Similarly subproblems force whole portions of the state space to follow the same subproblem solution. Therefore, these portions of the state space do not select their actions independently. One way to relax from statewise convexity to general convexity is to consider only a subset of stochastic games.

Theorem 5 *Consider no-control stochastic games, where all transitions are independent of the players' actions, i.e.,*

$$\forall s, s' \in \mathcal{S}\ \forall a, b \in \mathcal{A} \qquad T(s,a,s') = T(s,b,s').$$

If $\overline{\Pi}_i$ *is convex, then there exists a restricted equilibrium.*

Proof (Discounted Reward). This proof also makes use of Lemma 10, leaving us only to show that $\overline{\mathrm{BR}}_i(\pi_{-i})$ is convex. Just as in the proof of Theorem 4 we will consider the MDP defined for player i when the other players follow the fixed policy π_{-i}. As before it suffices to show that for this MDP, if $x_1, x_2 \in \overline{\Pi}$ are optimal for this MDP, then the policy $x_3(s,a) = \alpha x_1(s,a) + (1-\alpha)x_2(s,a)$ is also optimal for any $\alpha \in [0,1]$.

Again we use the notation $V^\pi(s)$ to refer to the traditional value of a policy π at state s in this fixed MDP. Since $T(s,a,s')$ is independent of a, we can simplify the Bellman equations (Equation 2.2) to,

$$\begin{aligned} V^x(s) &= \sum_a x(s,a)R(s,a) + \gamma \sum_{s'} \sum_a x(s,a)T(s,a,s')V^x(s') && (5.11) \\ &= \sum_a x(s,a)R(s,a) + \gamma \sum_{s'} T(s,\cdot,s')V^x(s'). && (5.12) \end{aligned}$$

For the policy x_3, the value of state s is then,

$$\begin{aligned} V^{x_3}(s) &= \alpha \sum_a x_1(s,a)R(s,a) + (1-\alpha)\sum_a x_2(s,a)R(s,a) + \\ &\quad \gamma \sum_{s'} T(s,\cdot,s')V^{x_3}(s'). && (5.13) \end{aligned}$$

Using equation 5.12 for both x_1 and x_2 we get,

$$\begin{aligned} V^{x_3}(s) &= \alpha(V^{x_1}(s) - \gamma \sum_{s'} T(s,\cdot,s')V^{x_1}(s')) + \\ &\quad (1-\alpha)(V^{x_2}(s) - \gamma \sum_{s'} T(s,\cdot,s')V^{x_2}(s')) + \\ &\quad \gamma \sum_{s'} T(s,\cdot,s')V^{x_3}(s') && (5.14) \\ &= \alpha V^{x_1}(s) + (1-\alpha)V^{x_2}(s) + \\ &\quad \gamma \sum_{s'} T(s,\cdot,s')\left(V^{x_3}(s') - \alpha V^{x_1}(s') - (1-\alpha)V^{x_2}(s')\right) && (5.15) \end{aligned}$$

Notice that a solution to these equations is $V^{x_3}(s) = \alpha V^{x_1}(s) + (1-\alpha)V^{x_2}(s)$, and the Bellman equations must have a unique solution. Hence, $V^{x_3}(s_0)$ is equal to $V^{x_1}(s_0)$ and $V^{x_2}(s_0)$, which are equal since both are optimal. So x_3 is optimal, and $\overline{\mathrm{BR}}_i(\pi)$ is convex. Applying Lemma 10 we get that restricted equilibria exist. □

Proof (Average Reward). An equivalent definition to Equation 2.6 of a policy's average reward is,

$$V_i^\pi(s) = d^\pi(s) \sum_a \pi(s,a)R(s,a), \tag{5.16}$$

where $d^\pi(s)$ defines the distribution over states visited while following π after infinite time. For a stochastic game or MDP that is unichain, we know that this distribution is independent of the initial state. In the case of no-control stochastic games or MDPs, this distribution becomes independent of the actions and policies of the players, and depends solely on the transition probabilities. So Equation 5.16 can be written,

$$V_i^\pi(s) = d(s) \sum_a \pi(s,a) R(s,a). \tag{5.17}$$

As before, we must show that $\overline{\mathrm{BR}}_i(\pi_{-i})$ is convex to apply Lemma 10. Consider the MDP defined for player i when the other players follow the policy π_{-i}. It suffices to show that for this MDP, if $x_1, x_2 \in \overline{\Pi}$ are optimal for this MDP, then the policy $x_3(s,a) = \alpha x_1(s,a) + (1-\alpha)x_2(s,a)$ is also optimal for any $\alpha \in [0,1]$. Using Equation 5.17, we can write the value of x_3 as,

$$\begin{aligned}
V_i^{x_3}(s) &= d(s) \sum_a x_3(s,a) R(s,a) && (5.18)\\
&= d(s) \sum_a \left(\alpha x_1(s,a) + (1-\alpha) x_2(s,a)\right) R(s,a) && (5.19)\\
&= d(s) \left(\sum_a \alpha x_1(s,a) R(s,a) + \sum_a (1-\alpha) x_2(s,a) R(s,a) \right) && (5.20)\\
&= \alpha \left(d(s) \sum_a x_1(s,a) R(s,a) \right) + (1-\alpha) \left(d(s) \sum_a x_2(s,a) R(s,a) \right) && (5.21)\\
&= \alpha V_i^{x_1}(s) + (1-\alpha) V_i^{x_2}(s). && (5.22)
\end{aligned}$$

Therefore x_3 has the same average reward as x_1 and x_2 and so is also optimal. So $\overline{\mathrm{BR}}_i(\pi_{-i})$ is convex and by Lemma 10 there exists an equilibrium. □

We can now merge Theorem 4 and Theorem 5 allowing us to prove existence of equilibria for a general class of games where only one of the player's actions affects the next state.

Theorem 6 *Consider single-controller stochastic games (Filar & Vrieze, 1997), where all transitions depend solely on player 1's actions, i.e.,*

$$\forall s, s' \in \mathcal{S}\ \forall a, b \in \mathcal{A} \qquad a_1 = b_1 \Rightarrow T(s,a,s') = T(s,b,s').$$

In the discounted reward formulation, if $\overline{\Pi}_1$ is statewise convex and $\overline{\Pi}_{i \neq 1}$ is convex, then there exists a restricted equilibrium.

Proof. This proof again makes use of Lemma 10, leaving us to show that $\overline{\mathrm{BR}}_i(\pi_{-i})$ is convex. For $i = 1$ we use the argument from the proof of Theorem 4. For $i \neq 1$ we use the argument from Theorem 5. □

The previous results have looked at stochastic games whose transition functions have particular properties. Our final theorem examines stochastic games where the rewards have a particular structure. Specifically we address team games, where the agents all receive equal payoffs.

Theorem 7 *For team games, i.e.,*

$$\forall i, j \in \{1, \ldots, n\}\ \forall s \in \mathcal{S}\ \forall a \in \mathcal{A} \qquad R_i(s,a) = R_j(s,a),$$

there exists a restricted equilibrium.

Proof. The only constraints on the players' restricted policy spaces are those stated at the beginning of this section: non-empty and compact. Since $\overline{\Pi}$ is compact, being a Cartesian product of compact sets, and player one's value in either formulation is a continuous function of the joint policy, then the value function attains its maximum (Gaughan, 1993, Corollary 3.11). Specifically, there exists $\pi^* \in \overline{\Pi}$ such that,

$$\forall \pi \in \overline{\Pi} \qquad V_1^{\pi^*} \geq V_1^{\pi}.$$

Since $V_i = V_1$ we then get that the policy π^* maximizes all the players' rewards, and so each must be playing a restricted best-response to the others' policies. □

5.2.3 Summary

Facts 1 and 5 provide counterexamples that show the threat limitations play to equilibria. Theorems 3, 4, 6, and 7 give us four general classes of stochastic games and restricted policy spaces where equilibria are guaranteed to exist. The fact that equilibria do not exist in general raises concerns about equilibria as a general basis for multiagent learning in domains where agents have limitations. On the other hand, combined with the model of implicit games, the presented theoretical results lay the initial groundwork for understanding when equilibria can be relied upon and when their existence may be in question. These contributions also provide some formal foundation for applying multiagent learning in limited agent problems.

5.3 Learning Restricted Equilibria

At the beginning of this chapter we highlighted the importance of the existence of equilibria to multiagent learning algorithms. This section presents results of applying a particular learning algorithm to a setting of limited agents. We use the best-response learner, WoLF-PHC (Bowling & Veloso, 2002a) introduced in Chapter 4. Recall that this algorithm is rational, that is, it is guaranteed to converge to a best-response if the other players' policies converge. In addition, it has been empirically shown to converge in self-play, where both players use WoLF-PHC for learning and the players are not limited. In this section we apply this algorithm in self-play to repeated games with limited players. Since the algorithm is rational, if the players converge their converged policies must be an equilibrium.

The specific limitations we examine fall into both the restricted policy space model as well as the implicit game model. One player is restricted to playing strategies that are the convex hull of a subset of the available strategies. From Theorem 3, there exists a restricted equilibrium with these limitations. For the limited player, the WoLF-PHC algorithm is modified slightly so that the player maintains Q-values of its restricted set of available strategies and performs its usual hill-climbing in the mixed space of these strategies. The unlimited player is unchanged and completely uninformed of the limitation of its opponent.

5.3.1 Rock-Paper-Scissors

The first game we examine is rock-paper-scissors. In Chapter 4 we presented results of WoLF-PHC applied to this matrix game. These results are shown again, in a slightly different format, in

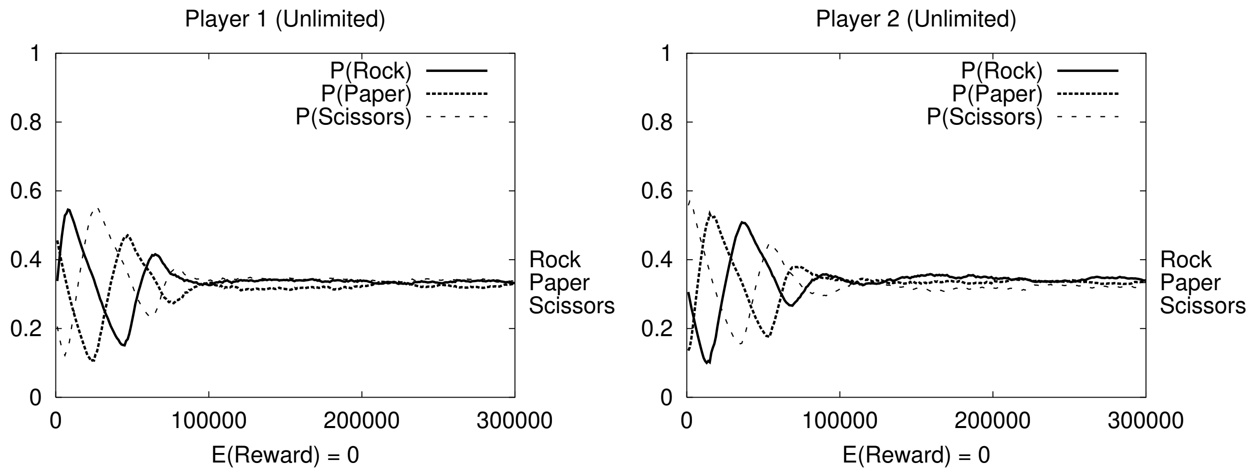


Figure 5.4: Rock-paper-scissors: strategy trajectories when learning with WoLF-PHC. Also see Figure 4.4.

Figure 5.4. Each graph shows the mixed policy the player is playing over time. The labels to the right of the graph signify the probabilities of each action in the game's unique Nash equilibrium. Observe that the players' strategies converge to this learning fixed point.

Figure 5.5 shows the results of restricting player 1 to a convex restricted policy space, defined by requiring the player to play "Paper" exactly half of the time. This is the same restriction as shown graphically in Figure 5.1. The graphs, again, show the players' strategies over time, and the labels to the right now label the game's restricted equilibrium, which accounts for the limitation (see Figure 5.1). The player's strategies now converge to this new learning fixed point. If we examine the expected rewards to the players, we see that the unrestricted player gets a higher expected reward in the restricted equilibrium than in the game's Nash equilibrium ($1/6$ compared to 0). In summary, both players learn optimal best-response policies with the unrestricted learner appropriately taking advantage of the other player's limitation.

5.3.2 Colonel Blotto

The second game we examined is "Colonel Blotto" from Figure 2.1(e). As a reminder, in this game, players simultaneously allot regiments to one of two battlefields. If one player allots more armies to a battlefield than the other, they receive a reward of one plus the number of armies the other player allotted to the battlefield, and the other player loses this amount. If the players tie, then the reward is zero for both. In the unlimited game, the row player has four regiments to allot, and the column player has only three.

Figure 5.6 shows experimental results with unlimited players. The labels on the right signify the probabilities associated with the Nash equilibrium to which the players' strategies converge. Player 1 is then given the limitation that it could only allot two of its armies, the other two would be allotted randomly. This is also a convex restricted policy space. Therefore by Theorem 3, it has a restricted equilibrium. Figure 5.7 shows the learning results. The labels to the right correspond to the action probabilities for the restricted equilibrium, which was computed by hand. As in rock-paper-scissors, the players' strategies converge to the new learning fixed point. Similarly, the

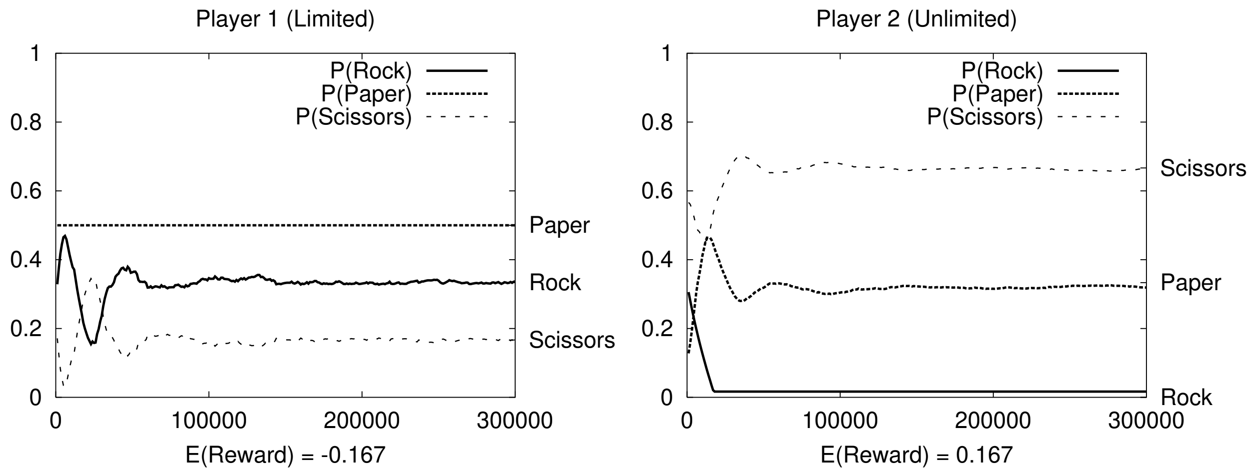


Figure 5.5: Rock-paper-scissors: strategy trajectories when learning with WoLF-PHC and Player 1 must play "Paper" with probability $\frac{1}{2}$.

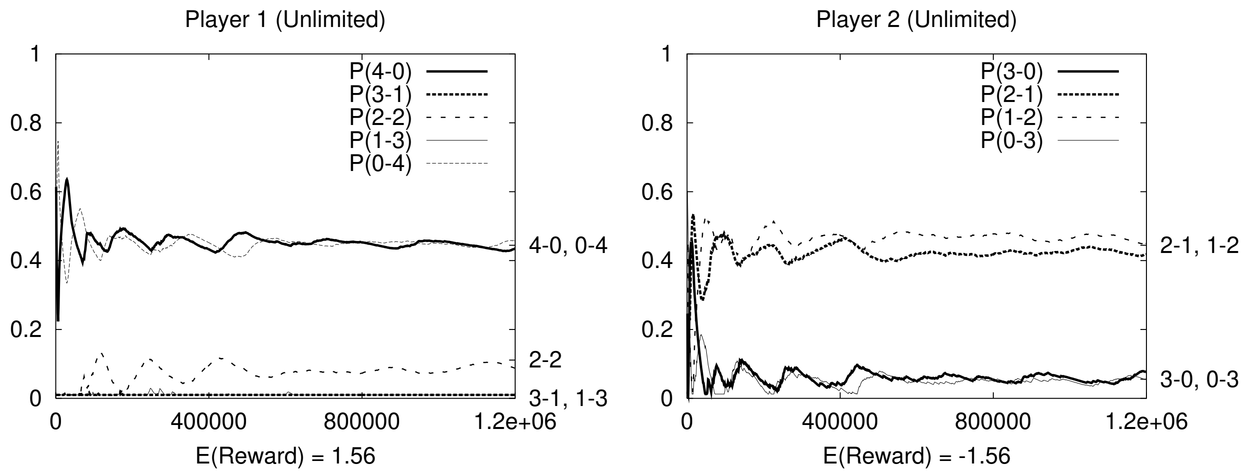


Figure 5.6: Colonel Blotto: strategy trajectories when learning with WoLF-PHC.

expected reward for the unrestricted player resulting from the restricted equilibrium is considerably higher than that of the Nash equilibrium (compare 0 to $-14/9$), as the player takes advantage of the other's limitation.

There is one final observations about these results. In Section 5 we discussed the use of rational limitations to speed learning. Even in these very small single-state problems, our results demonstrate that limitations can be used to speed learning. Notice that convergence occurs more quickly in the limited situations where one of the players has less parameters and less freedom in its policy space. In the case of the Colonel Blotto game this is a dramatic difference. (Notice the x-axes differ by a factor of four!) In games with very large state spaces this will be even more dramatic. Agents will need to make use of rational limitations to do any learning at all, and similarly the less restricted agents will likely be able to benefit from taking advantage of the more limited learners [5].

[5]Strictly speaking, restricted agents are not always at a disadvantage. In zero-sum games, such as those for which results are presented here, and team-games where all players rewards are identical, restrictions can never benefit the

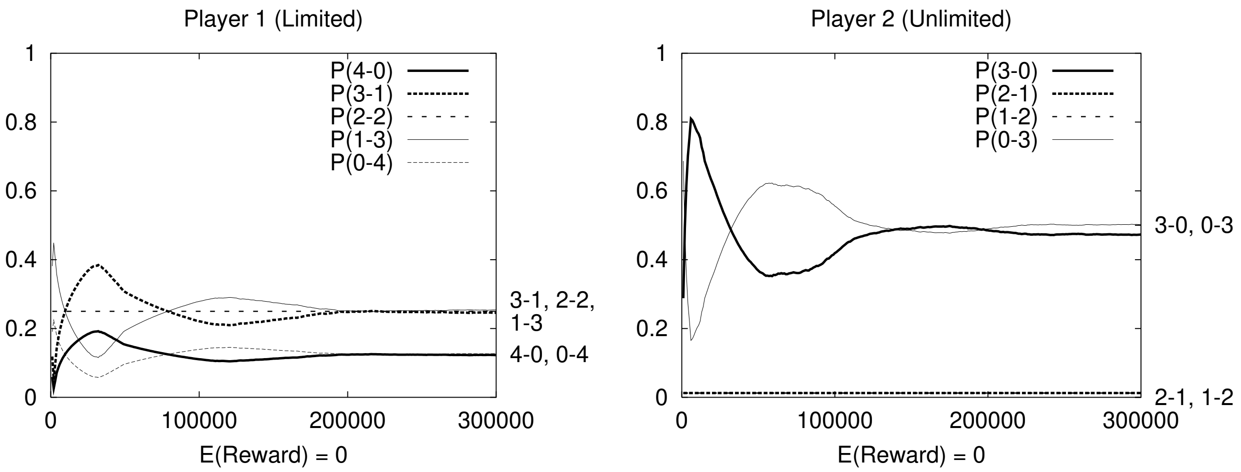


Figure 5.7: Colonel Blotto: strategy trajectories when learning with WoLF-PHC and Player 1 is forced to randomly allot two regiments.

5.4 Summary

We analyzed the effect of limitations on the Nash equilibrium solution concept in stochastic games. We investigated sources of different limitations and defined the model of restricted policy spaces as a generalization of limitations effects on agent behavior. We then proposed restricted equilibria as a solution concept that accounts for agent limitations and explored when restricted equilibria can be guaranteed to exist. In general, even with well-behaved agent limitations, restricted equilibria cannot be guaranteed to exist. We did define several general classes of stochastic games and restricted policy spaces where equilibria are guaranteed to exist. We finally examined learning of restricted equilibria. We demonstrated empirically that the WoLF-PHC algorithm is capable of converging to restricted equilibria in matrix games, when they exist.

agent. In general-sum games, such as Bach or Stravinsky from Table 2.1(d), this is not true. In this case, an imposed restriction, such as the row player being forced to select its first action, can actually improve the agent's equilibrium value. See (Gilboa & Samet, 1989; Littman & Stone, 2001).

Chapter 6

GraWoLF: Learning with Limitations

In realistic and complex domains, agent limitations are unavoidable. Tasks with intractably large or continuous state spaces require the imposing of limitations to make learning even possible. In Chapter 5 we explored the effect of limitations on the Nash equilibrium concept. In this Chapter we examine the effect of limitations on learning. Although, equilibria are not guaranteed to exist when agents are limited, we still want agents that can learn when they or others are limited.

We introduce GraWoLF[1] as a general-purpose, scalable, multiagent learning algorithm. GraWoLF combines policy gradient techniques developed for approximate single-agent learning, with the WoLF multiagent learning principle to account for other simultaneously learning agents. The algorithm learns within the confines of a limited space of parameterized policies, and incorporates WoLF into this best-response learner. The variable learning rate accounts for the other learning agents with the possibility of exploiting their limitations (recall the simple examples of this exploitation from Section 5.3). This combination allows GraWoLF to handle both its own and others' limitations.

We begin by presenting two very different domains for which we have applied the GraWoLF: goofspiel and keepout. Goofspiel is a card game with an intractably large state space. Keepout is an adversarial robot task where we examine both learning on physical robots, as well as experiments in simulation. After presenting these complex domains, we introduce the GraWoLF algorithm. After describing GraWoLF in detail, we also describe tile coding, a method for approximating large or continuous state spaces. We use tile coding to apply GraWoLF to our two tasks. Before discussing the application and results of applying GraWoLF to these problems we examine issues of evaluation. We then show compelling results of GraWoLF learning simultaneously both in goofspiel, keepout with simulated robots, and keepout with the robots themselves. The goal of this evaluation is to examine whether GraWoLF is capable of learning in these challenging domains, and how WoLF affects learning.

6.1 Two Complex Domains

Later in this chapter we present experiments of applying GraWoLF to two complex tasks. We begin by presenting these two tasks as motivation for the development of the GraWoLF algorithm.

[1] GraWoLF is short for "Gradient-based Win or Learn Fast", and the 'a' has the same sound as in "gradient".

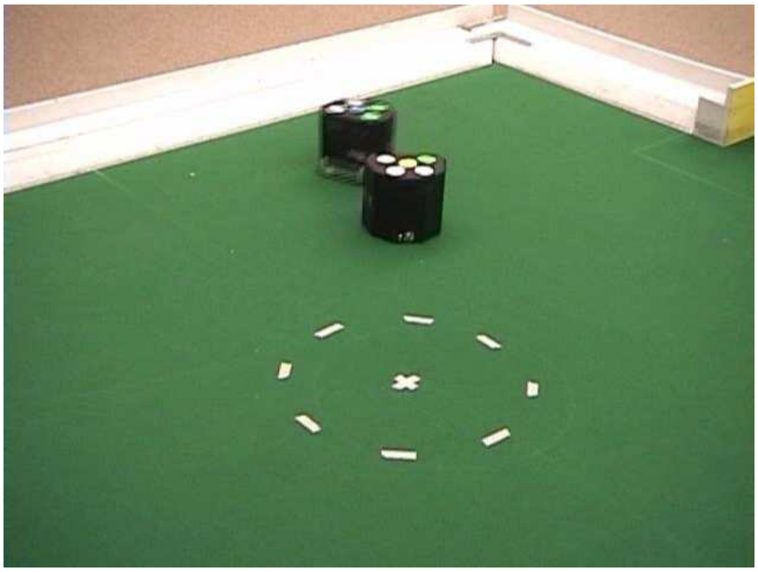


Figure 6.1: Keepout: an adversarial robot task. The top robot is trying to get inside the circle while the bottom robot is trying to stop it.

Following the two domain descriptions, we give a brief summary of the similarities and differences of these two domains.

6.1.1 Goofspiel

The first domain is goofspiel, the card game presented in Section 2.4.1 to illustrate the framework of stochastic games. It would be helpful to review the rules and details of the game that are presented in that section. In summary, two key points are: (1) the number of states of the game is far too large for learning without approximation. This requires agents to employ limitations to make learning possible. (2) The random policy is actually quite a good policy for this domain. We will use the random policy as the starting point for learning.

6.1.2 Keepout

Keepout is an adversarial robot task, and is shown in Figure 6.1. The robot at the top of the figure is the attacker and is trying to reach the circle in the center of the field. The robot closer to the circle is the defender and is trying to prevent this from happening. If the attacker reaches the circle, then it receives a reward of one and the defender receives a reward of negative one. These are the only rewards in the task. When the attacker reaches the circle or ten seconds elapses, the trial is over and the robots reset to their initial positions, where the attacker is a meter from the circle and the defender half-way between. The robots simultaneously learn in this environment each seeking to maximize its own discounted future reward. For all of our experiments the discount factor used was 0.8 for each full second of delay.

The robots themselves are part of the CMDragons '02 robot soccer team, which competes in the RoboCup small-size league. The complete team will actually be explored in depth in Chapter 7.

Here, we will only describe the details relevant to this learning task. First, the learning algorithm itself is situated within a large and complex architecture that already contains effective solutions for many aspects of the task. The team employs a global vision system mounted over the field. This input is processed by an elaborate tracking module that provides accurate positions and velocities of the robots. These positions and velocities comprise the input state for the learning algorithm. The team also uses robust modules for obstacle avoidance and motion control. The actions for the learning algorithm then involve providing target points for the obstacle avoidance module. Situating the learning within the context of this larger architecture focuses the learning. Rather than having the robot learn to solve well understood problems like path planning or object tracking, the learning is directed at the heart of the problem, the multi-robot interaction.

Second, the system control loop that is described above has inherent, though small, latency. Specifically, after an observed change in the world, 100ms will elapse before the robot's response is executed. This latency is overcome for the robot's own position and velocity by predicting through this latency period using knowledge of past, but not yet executed, actions. Since the actions of the opponent robots are not known, this prediction is not possible for other robots. Latency effectively adds an element of partial observability to the problem, since the agents do not have the complete state of the world, and in fact have separate views of this state. Notice, that this also adds a tactical element to successful policies. A robot can "fake" the opponent robot by changing its direction suddenly, since the other robot will not be able to respond to this change for a full latency period. Notice also that such movement must be stochastic, otherwise the opponent could predict the fake and respond prior to observing the change of direction.

Third, the system components not only add to the system latency, but they also cause further violations of the Markov property. Specifically, state-of-the-art solutions for vision and tracking, as well as fast path planning, involve maintaining state. For example, our navigation module employs Execution-extended Rapidly-exploring Random Trees (ERRTs) (Bruce & Veloso, 2002). This method uses waypoints from previously generated paths to bias path generation. The waypoint cache often has a large effect on the actual motion taken by the robot in navigating to a target point. This adds yet another element of partial observability to the task. It is certainly possible to expose the memory of the various subcomponents to the learning agent, but this would not solve the challenge entirely. First, the added input would cause a further state explosion since structures like a waypoint cache are extremely high-dimensional. Second, the waypoint cache (or similar such memory components) of the other agents is certainly not observable, and so this violation of the Markov property cannot be avoided.

In summary, this task involves numerous challenges. These challenges include continuous state and action spaces, elements of partial observability due to system latency, and violation of the Markov assumption since many of the system components have memory. None of these imposed limitations on agents match the criteria needed for equilibria to exist from Chapter 5.

6.1.3 Summary

These tasks are quite different but do have some similarities. Goofspiel is *fully observable* and *discrete*. The domain is fully observable since both players know the exact state of the game at any moment, i.e., there is no hidden information either in the state or in their hands. This is not to say it is a perfect information game, like Chess, because the players do not know the action the other player will select. Goofspiel has a discrete and finite (albeit very large) state space.

Keepout, however, is neither of these. It is *partially observable* and *continuous*. Latency results in each player knowing their own true position, but not their opponents. Technically, the players do not even know their own position exactly since their position is only a Kalman filtered estimate based on noisy observations. Partial observability extends further if one considers the state of memory-based components. Also unlike goofspiel, the state space is continuous and, therefore, infinite, which is an inherent challenge of any robot learning task.

The two domains do have a number of similarities. They are adversarial, intractable for state enumeration, and in both domains deterministic policies can be exploited by the other agent. Notice that exploitable deterministic policies is true for many multiagent tasks, e.g., rock-paper-scissors and Colonel Blotto from Section 2.3 and grid soccer from Section 2.4.1. In Section 2.4.1, we also described how deterministic policies in goofspiel can be soundly defeated by an opponent that always plays one higher than the player's deterministic policy. In comparison, the worst-case performance of the random policy is far better than that of the best deterministic policy. Keepout is similar. Since the defender always has less distance to travel to intercept the attacker, the attacker's only chance of success is to use system latency to its advantage. If the attacker changes directions, the defender will not be able to respond for a short time. If the attacker, though, changes directions in a deterministic way, the defender can anticipate and change directions simultaneously. Therefore the attacker necessarily must follow a stochastic policy to avoid being exploited.

6.2 Gradient-based WoLF

GraWoLF is a multiagent learning algorithm capable of scaling to complex problems such as goofspiel and keepout. The algorithm combines two powerful ideas. The first is the use of a parameterized policy and learning as gradient ascent in the policy's parameter space. The second is the use of a WoLF variable learning rate to adjust the gradient ascent step size. We will briefly overview these techniques and then describe how they are combined into a complete reinforcement learning algorithm. We finally describe a technique called tile coding for creating the feature vectors that are critical to GraWoLF. Tile coding is used in the application of this algorithm to goofspiel and keepout.

6.2.1 Policy Gradient Ascent

Policy gradient techniques (Williams & Baird, 1993; Sutton et al., 2000; Baxter & Bartlett, 2000) are a method of reinforcement learning with function approximation. Traditional approaches approximate a state-action value function, and result in a deterministic policy that selects the action with the maximum learned value. Alternatively, policy gradient approaches approximate a policy directly, and then use gradient ascent to adjust the parameters to maximize the policy's value. There are three good reasons for the latter approach. First, there is a whole body of theoretical work describing convergence problems using a variety of value-based learning techniques with a variety of function approximation techniques (see (Gordon, 2000) for a summary of these results). Second, value-based approaches learn deterministic policies, and as we mentioned earlier deterministic policies in multiagent settings are often easily exploitable[2]. Third, we have already demonstrated the successful application of gradient techniques for simultaneous learning in both matrix games and stochastic games with enumerable state spaces (see Chapter 4).

We use the policy gradient technique presented by Sutton and colleagues (Sutton et al., 2000). Specifically, we define a policy as a Gibbs distribution over a linear combination of features of a candidate state and action pair. Let θ be a vector of the policy's parameters and let ϕ_{sa} be an identically sized feature vector for state s and action a, then the Gibbs distribution defines a stochastic policy according to,

$$\pi_\theta(s,a) = \frac{e^{\theta \cdot \phi_{sa}}}{\sum_b e^{\theta \cdot \phi_{sb}}}.$$

Sutton and colleagues' main result was a convergence proof for the following policy iteration rule that updates a policy's parameters,

$$\theta_{k+1} = \theta_k + \delta_k \sum_s d^{\pi_k}(s) \sum_a \frac{\partial \pi_{\theta_k}(s,a)}{\partial \theta} f_{\mathbf{w}_k}(s,a). \tag{6.1}$$

For the Gibbs distribution this becomes,

$$\theta_{k+1} = \theta_k + \delta_k \sum_s d^{\pi_k}(s) \sum_a \phi_{sa} \Big(\pi_{\theta_k}(s,a) f_{\mathbf{w}_k}(s,a) \Big) \tag{6.2}$$

Here δ_k is an appropriately decayed learning rate. $d^{\pi_k}(s)$ is state s's contribution to the policy's overall value. This contribution is defined differently depending on whether average or discounted start-state reward criterion is used. For discounted reward with a start-state, which we denote, s_0,

$$d^\pi(s) \quad = \quad \sum_{t=0}^{\infty} \gamma^t Pr(s_t = s | s_0, \pi).$$

$f_{\mathbf{w}_k}(s,a)$ is an independent approximation of $Q^{\pi_k}(s,a)$ with parameters $\mathbf{w}$, which is the expected value of taking action a from state s and then following the policy π_k. For a Gibbs distribution, which is what we use in this work, Sutton and colleagues showed that for convergence this approximation should have the following form,

$$f_{\mathbf{w}_k}(s,a) \quad = \quad \mathbf{w}_k \cdot \left[\phi_{sa} - \sum_b \pi_{\theta_k}(s,b) \phi_{sb} \right]. \tag{6.3}$$

As they point out, this amounts to $f_{\mathbf{w}}$ being an approximation of the advantage function, $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, where $V^\pi(s)$ is the value of following policy π from state s. It is this advantage function that we estimate and use for gradient ascent.

Using this basic formulation we derive an on-line version of the learning rule, where the policy's weights are updated with each state visited. The summation over states can be removed by

[2]Deterministic policies are not a requirement for value-based approaches, yet any technique for mapping the learned values into a distribution over actions still cannot learn mixed equilibria. This is because, in the case of mixed equilibria, the value of all actions in the mixed strategy are the same. A mapping has no way of appropriately distinguishing between two games with very different mixed equilibria but identical equilibrium value.

updating proportionately to that state's contribution to the policy's overall value. Since we are visiting states on-policy then we only need to weight later states by the discount factor to account for their smaller contribution. If t time has passed since the trial start, this turns Equation 6.2 into,

$$\theta_{k+1} = \theta_k + \gamma^t \delta_k \sum_a \phi_{sa} \cdot \pi_{\theta_k}(s,a) f_{\mathbf{w}_k}(s,a). \tag{6.4}$$

Since the whole algorithm is on-line, we do the policy improvement step (updating θ) simultaneously with the value estimation step (updating $\mathbf{w}$). We do value estimation using gradient-descent Sarsa(λ) (Sutton & Barto, 1998) over the same feature space as the policy. This requires maintaining an eligibility trace vector, $\mathbf{e}$. The update rule is then, if at time k the system is in state s and takes action a transitioning to state s' after time Δt and then taking action a', we update the trace and the weight vector using,

$$\mathbf{e}_{k+1} = \lambda \gamma^{\Delta t} \mathbf{e}_k + \phi_{sa}, \tag{6.5}$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{e}_{k+1} \alpha_k \begin{pmatrix} r + \gamma^{\Delta t} Q_{\mathbf{w}_k}(s', a') \\ -Q_{\mathbf{w}_k}(s,a) \end{pmatrix}, \tag{6.6}$$

where λ is the Sarsa parameter and α_k is an appropriately decayed learning rate. The addition of raising γ to the power Δt allows for actions to take differing amounts of time to execute, as in semi-Markov decision processes (Sutton & Barto, 1998).

The policy improvement step then uses Equation 6.4 where s is the state of the system at step k and the action-value estimates from Sarsa, $Q_{\mathbf{w}_k}$, are used to compute the advantage term. We then get,

$$f_{\mathbf{w}_k}(s,a) = Q_{\mathbf{w}_k}(s,a) - \sum_a \pi_{\theta_k}(s,a) Q_{\mathbf{w}_k}(s,a). \tag{6.7}$$

This forms the crux of GraWoLF. What remains is the selection of the learning rate, δ_k. This is where the WoLF variable learning rate is used.

6.2.2 Win or Learn Fast

WoLF ("Win or Learn Fast") is a method for changing the learning rate to encourage convergence in a multiagent reinforcement learning scenario. Notice that the policy gradient ascent algorithm above does not account for a non-stationary environment that arises with simultaneous learning in stochastic games. All of the other agents actions are simply assumed to be part of the environment and unchanging. WoLF provides a simple way to account for other agents through adjusting how quickly or slowly the agent changes its policy.

As was discussed in its application to policy hill-climbing in Chapter 4, since only the rate of learning is changed, algorithms that are guaranteed to find (locally) optimal policies in non-stationary environments retain this property even when using WoLF. We have shown that in stochastic games with simultaneous learning but without limitations, WoLF has both theoretical and empirical evidence of it encouraging convergence in algorithms that do not otherwise converge. Recall that the intuition for this technique is that a learner should adapt quickly when it is doing more poorly than expected. When it is doing better than expected, it should be cautious, since the other players are likely to change their policy.

The WoLF principle naturally lends itself to be combined with policy gradient techniques where there is a well-defined learning rate, δ. With WoLF we replace the original learning rate with two learning rates, $\delta_w < \delta_l$, to be used when winning or losing, respectively. In Chapter 4 we used a notion of winning and losing by comparing the value of the current policy, $V^{\pi}(s)$, to the value of the average policy over time, $V^{\bar{\pi}}(s)$. So, if $V^{\pi}(s) > V^{\bar{\pi}}(s)$ then the algorithm is considered winning, otherwise it is losing. This was shown to be quite successful. With the policy gradient technique above we can define a similar rule that examines the approximate value, using Q_w, of the current weight vector, θ, with the average weight vector over time, $\bar{\theta}$. Specifically, we are "winning" if and only if,

$$\sum_a \pi_\theta(s,a) Q_w(s,a) > \sum_a \pi_{\bar{\theta}}(s,a) Q_w(s,a). \tag{6.8}$$

When winning in a particular state, we update the parameters for that state using δ_w, otherwise δ_l.

6.2.3 GraWoLF

GraWoLF is the combination of policy gradient ascent learning with WoLF. A feature vector over state and action pairs, ϕ_{sa}, is used for the parameterization of the policy and for the approximation of the policy's value. Gradient updates are then performed on both the value estimate using Equation 6.6 and the policy using Equation 6.4. WoLF is used to vary the learning rate δ_k in the policy update according to the rule in inequality 6.8. This composition can be essentially thought of as an actor-critic method (Sutton & Barto, 1998). Here the Gibbs distribution over the set of parameters is the *actor*, and the gradient-descent Sarsa(λ) is the *critic*. The WoLF principle is adjusting how the actor changes policies based on response from the critic. The complete details are shown in Table 6.1.

The missing ingredient for applying this algorithm to a particular task is the definition of ϕ_{sa}. This is task dependent and must define the critical features of a particular state and action. This is where approximation and generalization enter into the learning to make learning tractable. We now examine one particular method for generating feature vectors from a large or continuous state space. We use this technique for our learning in both goofspiel and keepout.

6.2.4 Tile Coding

Tile coding (Sutton & Barto, 1998), also known as CMACS (Albus, 1971), is a popular technique for creating a set of boolean features from a set of continuous features. In reinforcement learning, tile coding has been used extensively to create linear approximators of state-action values (e.g., (Stone & Sutton, 2001)).

The basic idea is to lay offset grids or tilings over the multidimensional continuous feature space. A point in the continuous feature space will be in exactly one tile for each of the offset tilings. Each tile has an associated boolean variable, so the continuous feature vector gets mapped into a very high-dimensional boolean vector. In addition, nearby points will fall into the same tile for many of the offset grids, and so share many of the same boolean variables in their resulting vector. This provides the important feature of generalization. An example of tile coding in a two-dimensional continuous space is shown in Figure 6.2. This example shows two overlapping tilings, and so any given point falls into two different tiles.

1. Let $\alpha \in (0, 1]$, $\delta^l > \delta^w \in (0, 1]$, and $\beta \in (0, 1]$ be learning rates. Initialize,
$$\mathbf{w} \leftarrow \vec{0} \qquad \theta \leftarrow \vec{0} \qquad \bar{\theta} \leftarrow \vec{0} \qquad \mathbf{e} \leftarrow \vec{0}$$

2. Define,
$$\begin{aligned}
\pi_\theta(s, a) &= \frac{e^{\theta \cdot \phi_{sa}}}{\sum_b e^{\theta \cdot \phi_{sb}}}, \\
Q_{\mathbf{w}}(s, a) &= \mathbf{w} \cdot \phi_{sa}, \\
f_{\mathbf{w}}(s, a) &= Q_{\mathbf{w}}(s, a) - \sum_a \pi_\theta(s, a) Q_{\mathbf{w}}(s, a).
\end{aligned}$$

3. Repeat,

   (a) From state s select action a according to mixed strategy $\pi^\theta(s)$.

   (b) Observing reward r and next state s' with elapsed time t,
   $$\begin{aligned}
   \mathbf{e} &\leftarrow \lambda \gamma^t \mathbf{e} + \phi_{sa} \\
   \mathbf{w} &\leftarrow \mathbf{w} + \alpha \left(r + \gamma Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a) \right) \\
   \theta &\leftarrow \theta + \gamma^t \delta \sum_a \mathbf{e} \cdot \pi_\theta(s, a) f_{\mathbf{w}}(s, a).
   \end{aligned}$$
   where,
   $$\delta = \begin{cases} \delta^w & \text{if } \sum_a \pi_\theta(s, a) Q_{\mathbf{w}}(s, a) > \sum_a \pi_{\bar{\theta}}(s, a) Q_{\mathbf{w}}(s, a) \\ \delta^l & \text{otherwise} \end{cases}.$$

   (c) Maintain an average parameter vector,
   $$\bar{\theta} \leftarrow (1 - \beta)\bar{\theta} + \beta\theta.$$

   (d) If s' is s_0 or the trial is over then $t \leftarrow 0$ and $\mathbf{e} \leftarrow \vec{0}$. Otherwise $t \leftarrow t + 1$.

Table 6.1: GraWoLF, a scalable e multiagent learning algorithm.

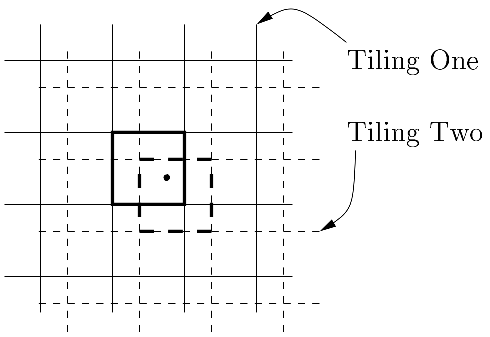


Figure 6.2: An example of tile coding a two dimensional space with two overlapping tilings.

Another common trick with tile coding is the use of hashing to keep the number of parameters manageable. Each tile is hashed into a table of fixed size. Collisions are simply ignored, meaning that two unrelated tiles may share the same parameter. Hashing reduces the memory requirements with little loss in performance. This is because only a small fraction of the continuous space is actually needed or visited while learning, and so independent parameters for every tile are often not necessary. Hashing provides a means for using only the number of parameters the problem requires while not knowing in advance which state-action pairs need parameters.

6.3 Evaluation

Evaluation is actually part of the ongoing research in the field of multiagent learning and is a difficult problem for a number of reasons. The standard single-agent method of evaluation is to compare an agent's performance over time with the value of optimal performance. Since stochastic games do not have optimal policies independent of the others' policies, such a comparison is not possible.

One straightforward evaluation technique for games is to have two learning algorithms learn against each other and simply examine the expected reward over time. This technique also has problems. First, this technique is not useful if one is interested in learning in self-play, where both players use an identical algorithm. In the case of a symmetric competitive game, such as goofspiel, the expected reward of the two agents is necessarily zero, providing no information. Secondly, other multiagent algorithms that were reviewed in Chapter 3 simply have not been applied to tasks as large and complicated as goofspiel and keepout. Making these algorithms applicable to such complex problems is a challenging problem in itself and well beyond the scope of this work.

Another common evaluation criterion is that of convergence. This is true in single-agent learning as well as multiagent learning. Convergence was also our focus in Chapter 4 where we examined convergence of joint policies to that of the equilibrium. Since equilibria requires that all players are doing as well as they can given the others' policies, this is a compelling result for the effectiveness of learning. Although convergence to an equilibrium is an ideal criterion for small problems, there are a number of difficulties with using this in evaluating learning with limitations. First, when we account for agent limitations then equilibria may cease to exist as was demonstrated in Chapter 5. This makes convergence of policies impossible. Second, policy gradient techniques learn only locally optimal policies. They may converge to policies that are not globally optimal and therefore necessarily not an equilibrium.

Although convergence to equilibria, and therefore convergence in general, is not a feasible criterion we still want learning agents to learn something in self-play. We propose and use three different evaluation techniques to examine learning in goofspiel and keepout: *challengers*, *improvement over the initial policy*, and *expected value during learning*. The first two techniques investigate the effectiveness of the policy being learned. Evaluation of a policy must always be with respect to some specific policy for the other agents, and as we show these two methods examine two different possible opponent policies for comparison. The final evaluation method examines the expected rewards to the players *while learning*. Although, convergence of policies may not be possible, actual rewards of the players may converge or at least be more or less stable.

Challengers. The evaluation technique of challengers was first used by Littman to evaluate Minimax-Q (Littman, 1994). We also used the technique in evaluating WoLF-PHC in Section 4.4.3. The idea is to evaluate the quality of a learned policy by examining its worst-case performance. We take an agent trained with another simultaneously learning agent, and then freeze its policy, and train a challenger to specifically defeat the learned policy. This technique can be viewed as measuring a number of different things. First, it measures the robustness of the learned policy to different opponents. Second, in unlimited adversarial settings it measures the closeness of the policy to the equilibrium. The policy that maximizes the performance against its challenger is part of an equilibrium to the game. Third, the challenger value is the performance that would be attained if the learning agent were to stop learning, letting the other agent continue learning. Obviously, higher values against its challenger is desirable.

Improvement over the initial policy. In evaluation using challengers, the learned policy is evaluated with respect to its worst-case performance, where policies are trained to defeat it. We can also examine the value of a learned policy against other policies. These comparisons will require some additional terminology. In two player games, value is determined by the two players' policies. When referring to a particular value, we will write it as "x v. y" where x and y will refer to a policy for player one and player two, respectively. There are three policies we will consider in this evaluation: **R**, **LL**, and **LR**. **R** simply refers to the random policy, i.e., the initial policy for the agents. **LL** refers to a policy trained in self-play against another learning agent. **LR** refers to a policy trained against another agent following the random policy, and so is not learning.

We want to examine the question of whether learning improved the agent's performance. There are two different ways of examining this question. One method is to consider whether the agent would prefer to switch back to its initial policy at the end of training. This amounts to comparing whether **LL** v. **LL** is greater than **R** v. **LL**. A second method is to consider the hypothetical outcome if the agent had not bother learning at all. This amounts to comparing whether **LL** v. **LL** is greater than **R** v. **LR**. Notice in the second case, comparison is to the value of the policy that the other agent would have learned if the agent had chosen not to learn.

These two comparisons, though, are actually redundant. One would expect that **R** v. **LR** is less than **R** v. **LL**, since the learning in the former case is specifically focused on defeating the random policy. So, if the first method of comparison were to hold, i.e., that **LL** v. **LL** is greater than **R** v. **LL**, then we get the following,

$$\mathbf{R}\text{ v. }\mathbf{LR} < \mathbf{R}\text{ v. }\mathbf{LL} < \mathbf{LL}\text{ v. }\mathbf{LL},$$

and so the second method of comparison holds as well. For this reason we focus on the first method of comparison in determining whether the agent's policy improved with learning.

Expected value during learning. The final technique is to examine the actual values of the policies being played during learning. Even if policies do not converge, expected rewards might converge. This evaluation technique is specifically targeting the question of whether learning oscillates as was observed in smaller problems in Section 4.4. This will also help isolate the effect of WoLF on these reward oscillations.

6.4 Results

Applying GraWoLF to a task involves finding an appropriate parameterization of the state and action space through the specification of the feature vector ϕ_{sa}. We have already described tile coding as a general method for doing this, but there are still many variables in its application. The method of tiling is very important to the overall performance of learning as it is a powerful bias on what policies can be learned. The major decisions to be made are how to represent the state as a vector of numbers, which of these are tiled together, and the size and number of tiles used. The first and third decision determines what states are distinguishable, and the second and third determines how generalization works across distinguishable states. Despite the importance of the tiling in both tasks, we essentially selected what seemed like a reasonable tiling, and used it throughout our results.

Our results in both tasks compare three different learning algorithms. The algorithms only differ in the learning rates used and are named WoLF, Fast, and Slow. WoLF is the algorithm as shown in Table 6.1. Slow and Fast are identical but do not vary the learning rate, instead fixing δ to be δ_w or δ_l, respectively. The goal of these learning experiments is to answer two questions.

1. Can GraWoLF learn effectively in these complex tasks?

2. What effect does the WoLF principle have on learning?

We answer the first question by simply looking at GraWoLF's performance primarily along the first and second dimensions of evaluation: challengers and improvement over the initial policy. We address the second question by comparing the results of using the WoLF variable learning rate with the Fast and Slow fixed rates, focusing on the first and third evaluation techniques: challengers and expected value during learning.

6.4.1 Goofspiel

We now look at applying GraWoLF to the task of goofspiel. We start with describing the details of how the feature vector, ϕ_{sa}, is computed for the particular task, and then show the results of learning.

Applying GraWoLF

We represent a set of cards, either a player's hand or the deck, by five numbers, corresponding to the value of the card that is the minimum, lower quartile, median, upper quartile, and maximum. This provides information as to the general shape of the set, which is what is important in goofspiel. The other values used in the tiling are the value of the card that is being bid on and the card corresponding to the agent's action. An example of this process in the 13-card game is shown in Table 6.2. These values are combined together into three tilings. The first tiles together the quartiles describing the players' hands. The second tiles together the quartiles of the deck with the card available and player's action. The last tiles together the quartiles of the opponent's hand with the card available and player's action. The tilings use tile sizes equal to roughly half the number of cards in the game with the number of tilings greater than the tile sizes to distinguish between any integer state values. Finally, these tiles were all then hashed into a table of size one million

My Hand	1	3	4	5	6	8	11	13
Opp Hand	4	5	8	9	10	11	12	13
Deck	1	2	3	5	9	10	11	12
Quartiles	×		×		×	×		×
Card	11							
Action	3							

$$\left.\right\} \begin{array}{c} \langle 1, 4, 6, 8, 13 \rangle, \\ \langle 4, 8, 10, 11, 13 \rangle, \\ \langle 1, 3, 9, 10, 12 \rangle, \\ 11, 3 \\ \Downarrow \text{(Tile Coding)} \\ \phi_{sa} \in \{0, 1\}^{10^6} \end{array}$$

Table 6.2: Representation of an example state-action pair using quartiles to describe the players' hands and the deck. These numbers are then tiled and hashed with the resulting tiles representing a boolean vector of size 10^6.

in order to keep the parameter space manageable. We do not suggest that this is a perfect or even good tiling for this domain, but as we will show the results are still very interesting.

Experiments

The exact details of the parameters and learning rates used for these experiments can be found in the appendix. Throughout training and evaluation runs, all agents followed an ϵ-greedy exploration strategy with $\epsilon = 0.05$. That is, with 5% probability a random action was taken instead of the selected action. This insured that the value of all actions would continuously be updated. The initial policies and values all begin with zero weight vectors. With a Gibbs distribution, zero weights correspond to the random policy, which as we have noted is a relatively strong policy. We now look at each method of evaluation in turn.

Challengers. In our first experiment we trained the learner in self-play for 40,000 games. After every 5,000 games we stopped the training and trained a challenger against the agent's current policy. The challenger was trained on 10,000 games using Sarsa(λ) gradient ascent with identical learning rate parameters. The two policies, the agent's and its challenger's, were then evaluated on 1,000 games to estimate the policy's worst-case expected value. This experiment was repeated thirty times for each algorithm.

The learning results averaged over the thirty runs are shown in Figure 6.3 for suit sizes of $n = 4$, 8, and 13. The baseline comparison is with that of the random policy, a competitive policy for this game. All three learners improve on this policy while training in self-play. The initial dips in the 8 and 13 card games are due to the fact that value estimates are initially very poor making the initial policy gradients not in the direction of increasing the overall value of the policy. It takes a number of training games for the delayed reward of winning cards later to overcome the initial immediate reward of winning cards now. Lastly, notice the effect of the WoLF principle. It consistently outperforms the two static step size learners. This is identical to the effect shown in Chapter 4 for stochastic games with enumerable state spaces and no limitations

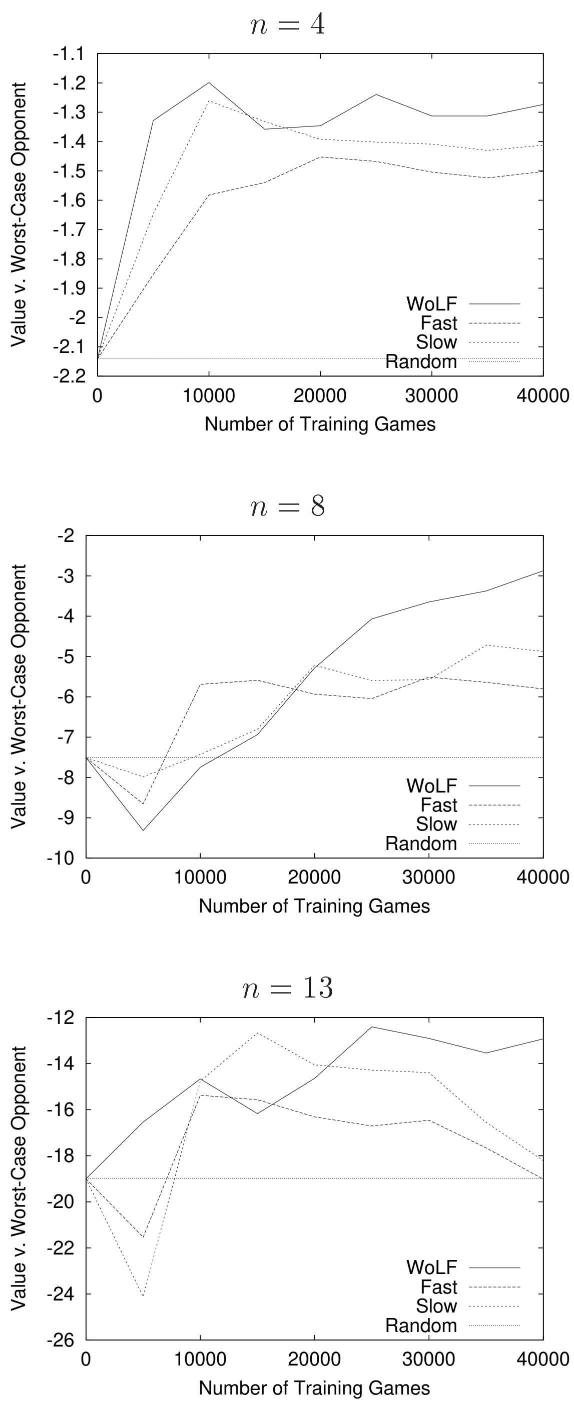


Figure 6.3: Goofspiel: results against challengers. The improvement of WoLF over random, Fast, and Slow after 40,000 training games is statistically significant for all three values of n at 95% confidence.

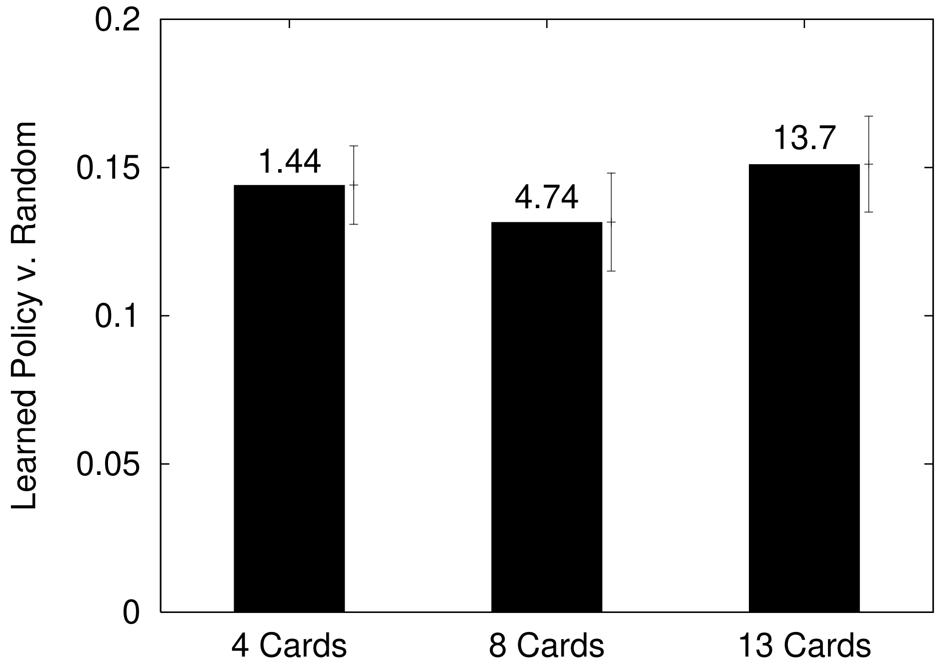


Figure 6.4: Goofspiel: Learned policies versus the random policy. The y-axis is the expected winnings as a fraction of the total points available in the game.

Improvement over the initial policy. The second experiment examines whether learning causes an improvement over the initial policy. As we described in Section 6.3, we simply compare **LL** v. **LL** to **R** v. **LL**, that is does learning improve the policy. Since goofspiel is a symmetric zero-sum game, the expected values of policies learned in self-play are necessarily zero. Hence, the desired improvement amounts to **R** v. **LL** being negative, which is the same as **LL** v. **R** being positive.

A policy was trained with GraWoLF in self-play over 40,000 games and then evaluated against the random policy for 1,000 games. The results averaged over thirty runs for all three suit sizes are shown in Figure 6.4. For all three sizes of games, the learned policies are significantly better than the random policy. This demonstrates that GraWoLF is making a distinct improvement over its initial random policy, which we have repeatedly noted is a relatively strong policy in this domain.

We can also examine how well the learned policy in self-play compares to the random policy *as learning progresses*. The value of the learned policy versus the random policy after every 5,000 trials of training in self-play are shown in Figure 6.5. Each point consists of 1,000 games of evaluation. It is interesting that the improvement over the random policy occurs very early on in training, and then the improvement levels off or diminishes. The eventual diminishing is not surprising since the agent being trained against begins playing the random policy, and so the initial trials are extremely relevant toward improving against that policy. What is interesting, though, is that performance against the random policy does not diminish a very large amount. This is true despite the fact that the training opponent is also learning and so is significantly different from the random policy after a few thousand trials. This suggests that, in addition to learning to optimize

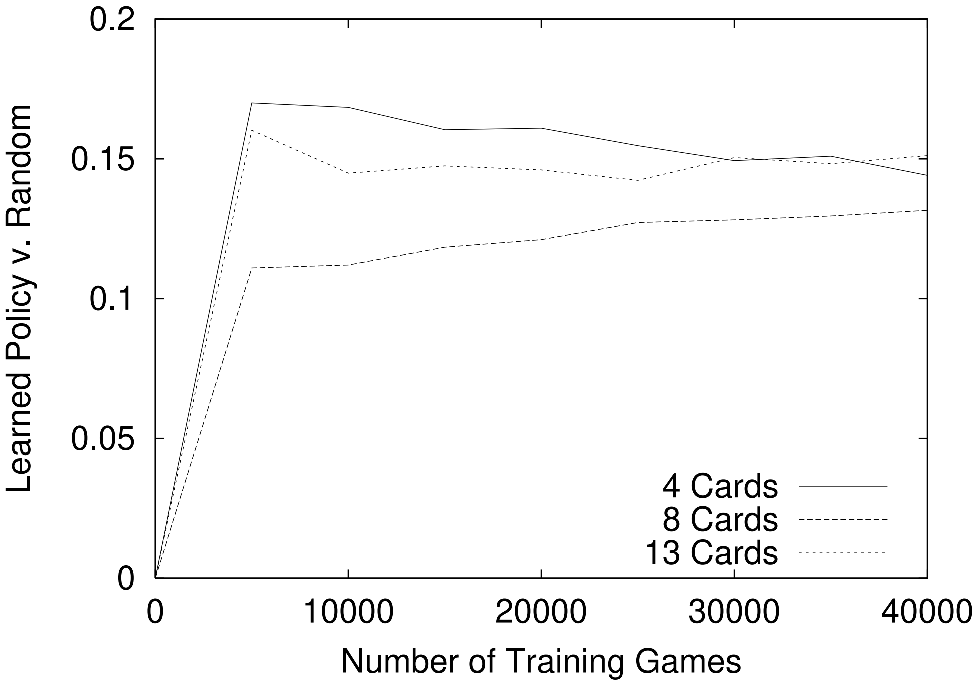


Figure 6.5: Goofspiel: Learned policies versus the random policy throughout learning. The y-axis is the expected winnings as a fraction of the total points available in the game. For comparison purposes, the numbers shown at the top of the bars are the absolute expected values of the learned policies versus the random policy.

against a particular opponent, the learned policy is still generally effective against other policies.

Expected value during learning. The third experiment examines stability of the player's actual expected value during learning, primarily examining the effect of the WoLF principle on the learning process. Again the algorithms were trained in self-play for 40,000 games. After every 50 games, both players' policies were frozen and evaluated over 1,000 games to find the expected value to the players of their policies at that moment. We ran each algorithm once on just the 13 card game and plotted its expected value over time while learning. The results are shown in Figure 6.6. Notice that the expected value of all the learning algorithms oscillates around zero. We would expect this with identical learners in a symmetric zero-sum game. The point of interest, though, is how close these oscillations remain around zero over time. The WoLF principle causes the policies to have a more stable expected value with lower amplitude oscillations when compared with the alternative of not using WoLF, i.e., using Fast or Slow. This again shows that the WoLF principle continues to have converging affects even in complex stochastic games involving limited agents.

6.4.2 Robot Keepout

We now turn our attention to keepout, the adversarial robot task. Our main interest is learning on the robot platform, but due to the tedium of repeated learning and evaluation we will also

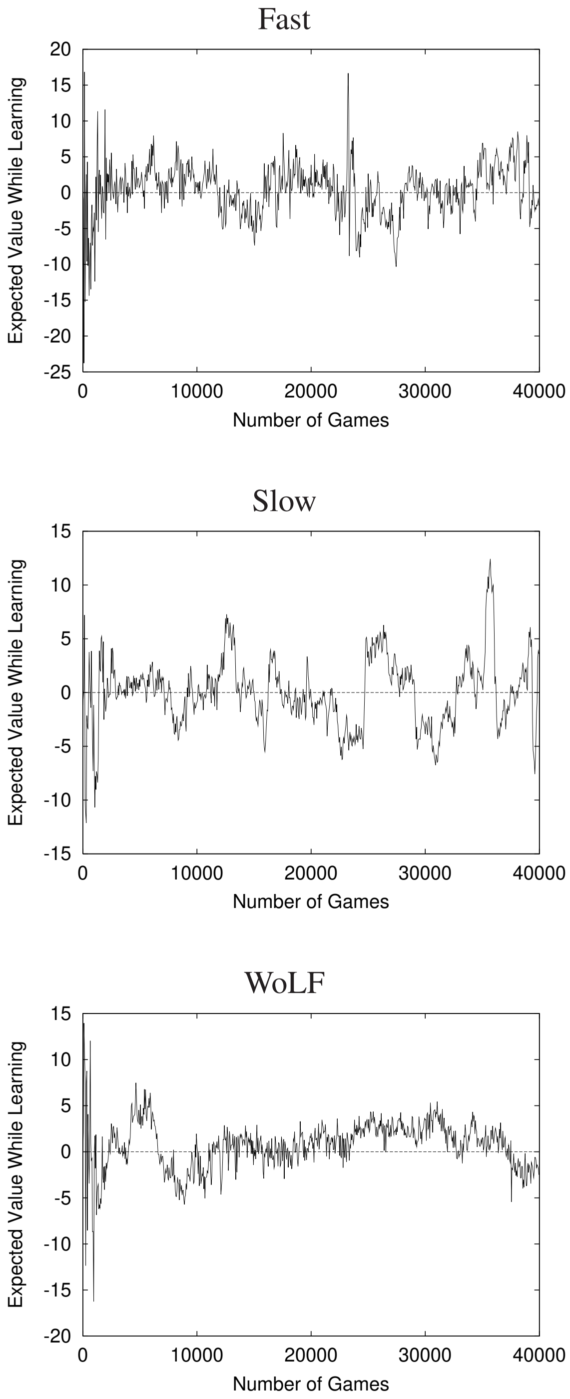


Figure 6.6: Goofspiel: Expected value of the game while learning.

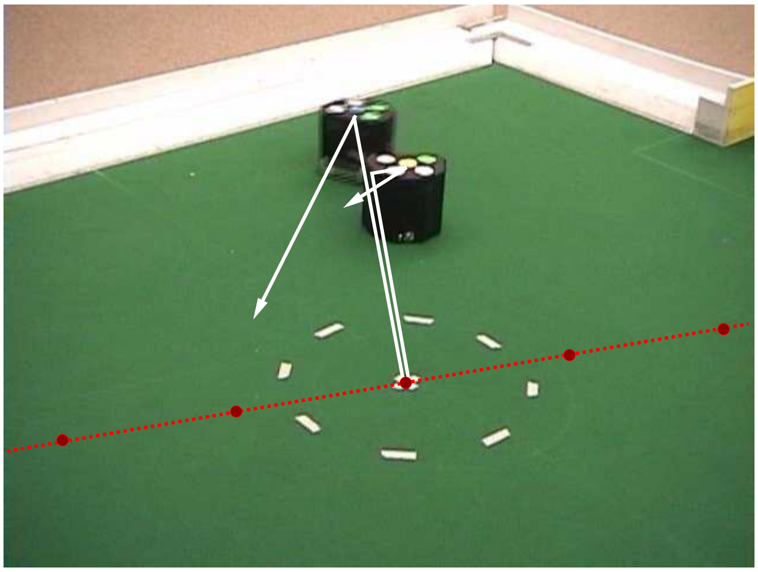


Figure 6.7: Keepout: an adversarial robot task. The top robot is trying to get inside the circle while the bottom robot is trying to stop it. The lines show the state and action representation.

explore this problem using a simulator of our robot team. This will allow us to gather statistically significant results and also to seed learning on the robot platform itself speeding up the training. We first discuss the application of GraWoLF to this task, and then show results of learning experiments in simulation and on the robot platform itself.

Applying GraWoLF

As with goofspiel the major detail to be decided in applying GraWoLF to the task is specifying the feature vector, ϕ_{sa}. In this task we have a continuous state and action space. The state consists of the Kalman filtered positions and velocities of the two robots, and the available actions are points on the field for navigation. By observing that the radial angle of the attacker with respect to the circle is not relevant to the task, we arrive at a seven dimensional input space. These seven dimensions are shown by the white overlaid lines in Figure 6.7.

We apply tile coding to this entire seven dimensional space. We use 16 tiles whose size is 800mm for distance dimensions and 1,600mm/s for velocity dimensions. We simplify the action space by requiring the attacker to select its navigation point along a perpendicular line through the circle's center. This is shown by the dark overlaid line in Figure 6.7. This line, whose length is three times the distance of the attacker to the circle, is then discretized into seven points evenly distributed. The defender uses the same points for its actions but then navigates to the point that bisects the line from the attacker to the selected action point. The robot's action is also included in the tiling as an eighth dimension with the tile size along that dimension being the entire line. Hence actions are distinguishable, but there is also a great deal of generalization. Agents select actions every tenth of a second, i.e., after every three frames, unless the feature vector has not changed over that time. This keeps the robots from oscillating too much during the initial parts of learning.

Experiments

We evaluate learning both in simulation and on the robot platform. We used a simulator specifically developed for our CMDragons robot soccer team. The simulator captures all of the important features of our team including the critical component of latency. One drawback of the version of the simulator used is its lack of a collision model. Although drastic collisions are rare in this domain since both robots are actively avoiding obstacles in their navigation, small collisions still happen. In order to allow for learning in simulation, the task was modified slightly. If the robots "collide" in simulation with the defender between the attacker and the goal, then the trial is considered a failure for the attacker, which receives no reward, and is ended. This same rule was employed in learning on the robot platforms but only came into play on rare serious collisions. The result of this modification is that the attacker is at a larger disadvantage in the simulator since trial-ending collisions are more common. This will be noticeable in comparing the relative success of the attacker in simulation versus the robot platform.

The exact details of the learning rates and parameters used for these experiments can be found in the appendix. As with goofspiel, the initial policies and parameters all begin with zero weight vectors, which corresponds to the random policy. In all of the performance graphs, the y-axis shows the attacker's expected discounted reward. A high value means the attacker is scoring in a short period of time and so the attacker is doing well. The range of expected times to score for any graph is shown to the right of the graph. All measurements of expected discounted reward are gathered empirically over the course of 1,000 trials. All training occurred over 4,000 trails, and takes approximately 6-7 hours of training time on the robot platform or in simulation, which runs in real time. Unless otherwise noted, the training was always done with a simultaneously learning opponent. The experiments in simulation were repeated ten times and the averages are shown in the graphs.

Unlike goofspiel, keepout is not a symmetric domain, and therefore requires comparisons of both policies learned by the attacker and by the defender. Each of our comparisons are repeated for both players in the task. All values presented, though, will be in terms of the attacker's expected reward. Hence for the defender, low values are good and correspond to the attacker doing poorly and the defender doing well.

Keepout is also considerably more complex than goofspiel, involving continuous state, partial observability, non-Markovian transitions, and for non-simulation results, robot noise. All of these are by themselves very challenging and at the edge of behavior learning research. Combining all of these in an adversarial multiagent task with a comparably impoverished amount of training, makes this a truly monumental task. Our results in this domain are, as a whole, quite compelling, but the various methods of evaluation are not as clearly convincing as the results presented in goofspiel. For completeness we present all of the results, although for some methods of evaluation, few or no conclusions may be drawn. We now look at each method of evaluation in turn.

Challengers. The challengers experimental setup is similar to goofspiel, although differs slightly due to the expense of gathering data and due to asymmetry in the task. Attacker and defender policies are trained using simultaneous learning. The policy is then fixed and a challenger policy is trained for 4,000 trials. The challenger policy learns using policy gradient ascent as presented in Table 6, but does not vary the learning rate, since its opponent is stationary. It instead uses a fixed

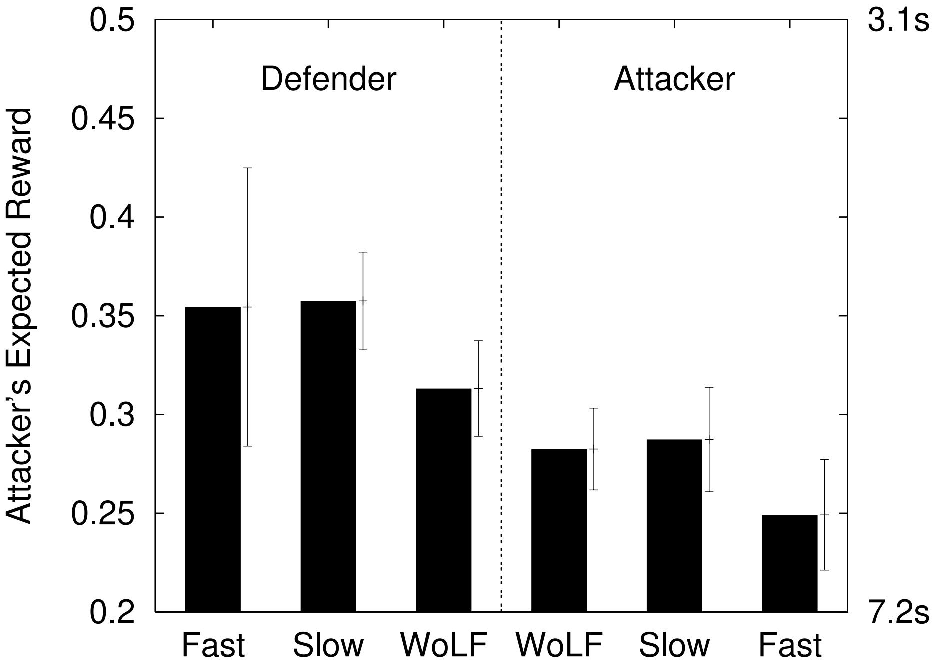


Figure 6.8: Keepout: Results against challengers in simulation.

rate equal to δ_l. The simultaneous learning was stopped after 3,000, 3,500, and 4,000 trials and a challenger was trained, and the resulting worst-case performances averaged together. This allowed more data to be gained from each training run. Figure 6.8 shows results averaged over ten runs.

The left side of the graph shows the performance of the learned defender policies against their challengers, while the right side shows the performance for attacker policies against their challengers. As with goofspiel, WoLF varies the learning rate as presented in Table 6, and Slow and Fast use a fixed learning rate of δ_w and δ_l, respectively. Notice that since the left side of the graph shows results for the defender policies, *low is better*, while for the right side, *higher is better*.

The defender policies trained in self-play using the WoLF variable learning rate have statistically significant more robust policies then using a fixed learning rate, either Fast or Slow. The learned attacker policy, on the other hand, is only a statistically significant improvement over the Fast learning rate against its challenger. The Slow learning rate has an indistinguishable effect from using WoLF when playing its challenger over the short training period.

There are a couple of possible explanations of this result. First, it may simply be that the random policy for the attacker is such a strong policy that the the use of the slow learning rate doesn't result in too large a deviation from it. In the next section, we give some evidence for this possibility. A second cause may be the particular initial values used. If we examine how often the slow and fast learning rates are used, we find that the attacker uses its winning (slow) learning rate for 92% of its updates, while the defender only uses it for 85%. Such a difference may be due to the task definition and initialization of the learner's values. For the defender, initial value estimates of zero amounts to an optimistic assessment, since the rewards to the defender are non-positive. For the attacker, the zero initialization amounts to a pessimistic assessment, since all rewards are non-negative. Pessimistic initial values would make the agent seem to be winning until these values were overcame. This would cause the attacker to mostly use the slower learning rate, keeping it from learning a better policy in the short learning period. Initial value estimates have an important effect on single-agent learning, (see (Brafman & Tennenholtz, 2002b) for theoretical results and further references to empirical results). Similar effects in the simultaneous learning setting is, therefore, not surprising, and deserves further investigation.

Improvement over the initial policy. Since keepout is an asymmetric game, we will need to compare both improvements in the attacker's, as well as the defender's, policies. This amounts to checking the following desired values,

$$\mathbf{LL} \text{ v. } \mathbf{R} > \mathbf{LL} \text{ v. } \mathbf{LL} > \mathbf{R} \text{ v. } \mathbf{LL}.$$

Another interesting comparison, since keepout is an asymmetric game, is to view the attacker and defender as a combined team and then consider whether learning in self-play improved the policies as a team. This comparison is similar to Swiss team tournaments in duplicate bridge, which due to the random hands is also an asymmetric game. To compare a pair of policies, **A** and **D**, with another pair of policies, **A'** and **D'**, one simply compares **A** v. **D'** with **A'** v. **D**. If the former is larger than the latter, then the original pair is superior. In keepout, we will use the learned policies from self-play as a team and compare with the initial random policies. The learned policies, then, are superior if **LL** v. **R** is greater than **R** v. **LL**.

Figure 6.9 shows the results of the values of the three key pairs of policies averaged over ten trials: **LL** v. **R**, **LL** v. **LL**, and **R** v. **LL**. First, notice that the first half of the inequality holds, **LL** v. **R** > **LL** v. **LL**. The second half of the inequality does not, although the difference is not statistically significant. This suggests that although the defender certainly prefers learning over not learning, the attacker is more ambivalent. This gives further evidence that the random policy for the attacker is quite a good policy to begin with, even against a strong defending policy. Using the Swiss team comparison, there is a dramatic improvement of learning for the combined policy, since **LL** v. **R** significantly outperforms **R** v. **LL**. These combined results suggest that the learning has a dramatic improvement on the defender's policy, while having only little effect on the attacker's policy. Again, this marginal improvement may be due to the random policy being a strong policy to begin with.

Expected value during learning. Our final evaluation tool is to examine the expected value during learning. Rather than freezing policies while learning and performing a very time-consuming computation of expected value, we simply averaged the actual value received over 50 trial blocks. To reduce noise, this was averaged over ten separate learning runs. These results for all three algorithms are shown in Figure 6.10. These results are shown mainly for the purpose of completeness and we do not attempt to draw any conclusions from these results.

Robot Learning. Finally, we examine results of using GraWoLF on the real robots. We begin with policies that were trained for 2000 trials in simulation and then perform an additional 2000 trials of training on the robots, all with simultaneous learning. Challenger evaluations are far too time-consuming to perform on the robot platform, so evaluation was restricted to examining improvement over the initial policy. As before, this was done by comparison to the random policy.

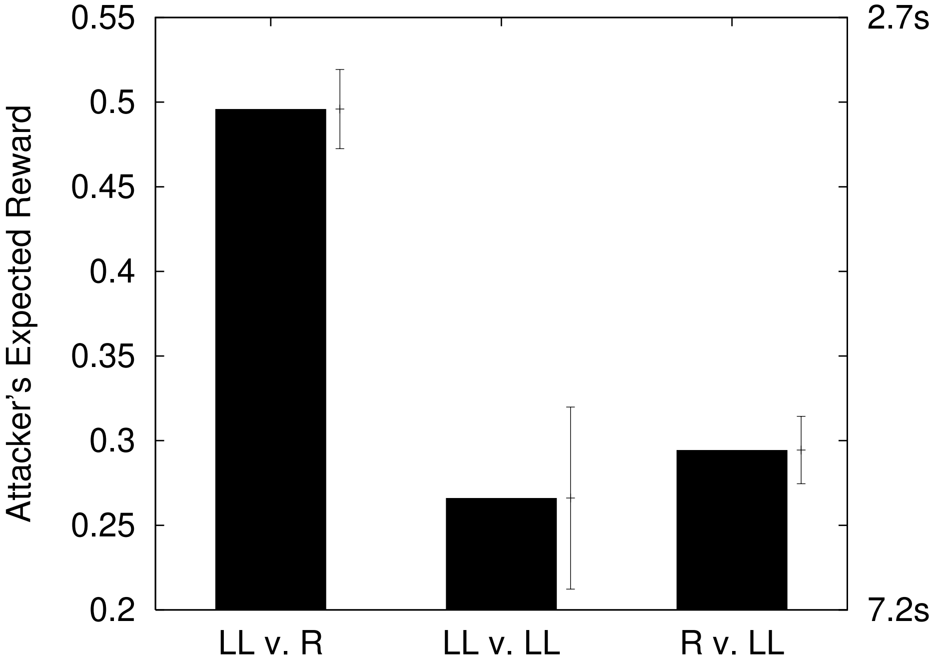


Figure 6.9: Keepout: Learned policies versus the random policy.

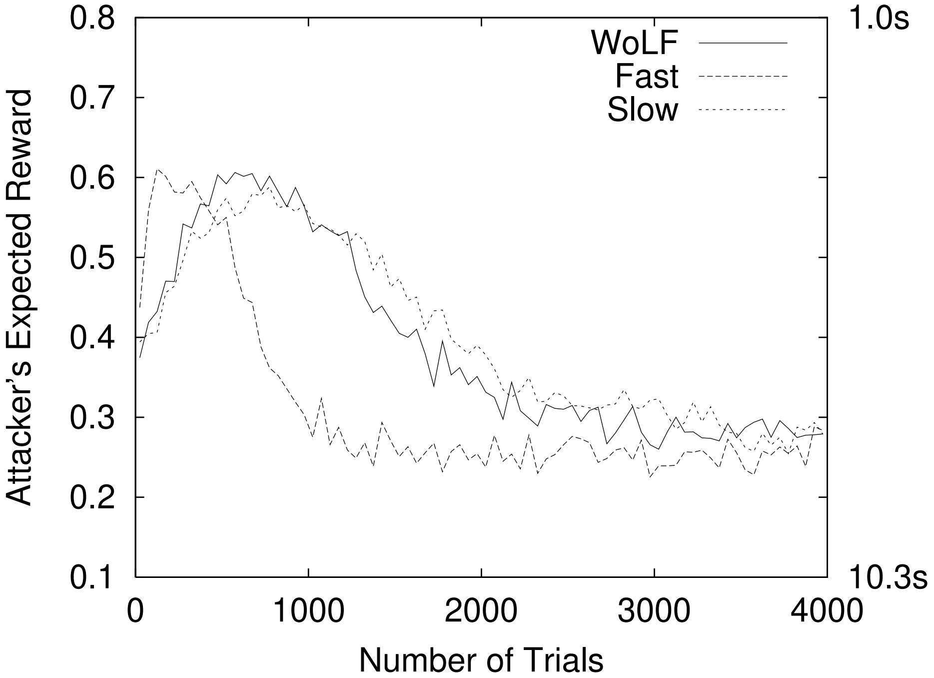


Figure 6.10: Keepout: Expected value of policies while learning.

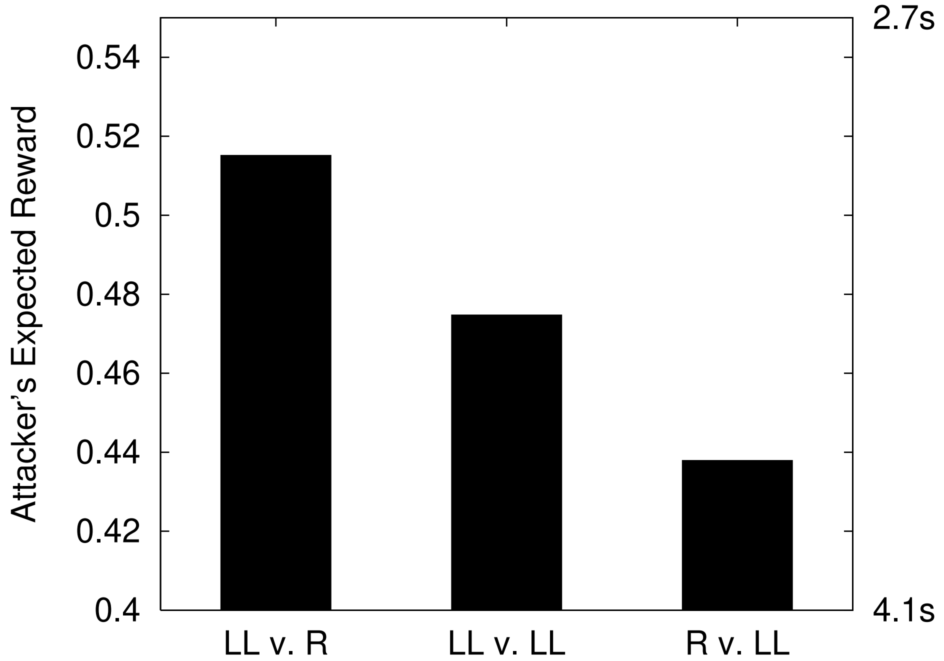


Figure 6.11: Keepout: Learned policies versus the random policy on the robot platform. Training was done with 2,000 trials in simulation and 2,000 trials on the robots.

The results of a single training run followed by 500 trials of evaluation are shown in Figure 6.11. The **LL** v. **LL** value was computed using the last 500 trials of learning, while the other values were experimentally obtained. These results are nearly identical to the simulation results in Figure 6.9. The desired inequality holds and the Swiss team comparison shows the learned policies are superior to the initial policies. All of these results are statistically significant for this single run.

Learned Policies. Since keepout is an adversarial environment, we would expect good policies to be stochastic in nature. This is true of all of the learned policies in both simulation and on the robots. For example, the most probable action in any state visited has probability on average around 40% in the attacker's learned policy, and 70% in the defender's. In some states, though, the learned policies are nearly deterministic. Hence, the learned policies are indeed stochastic to avoid exploitation, while still learning optimal actions in states that do not depend on the other agent's behavior.

Finally, one of the most compelling result of these experiments is actually observing the learned policies. This is especially true for the policies learned on the robots themselves. Not only is it obvious that the learned policy is stochastic, but it can also be observed that the attacker does indeed exploit the defender's latency to "fake" the opponent in order to reach the circle. In addition, the defender is forced to guess the attacker's change of direction in order to compensate. A video of the resulting policies running on the robots can be found at the following URL, "http://www.cs.cmu.edu/˜mhb/research".

6.4.3 Conclusion

These results, in both goofspiel and keepout, suggest that GraWoLF is capable of learning effective policies in multiagent simultaneous learning tasks. These policies are both robust in the worst-case and an improvement over the initial policy. In addition, the WoLF variable learning rate has the same powerful effect observed in the results in Chapter 4. It appears to dampen oscillation and guide the learning toward robust, near equilibrium (if they exist) policies. This was demonstrated on two very different, yet very challenging, tasks.

6.5 Summary

In this chapter we introduced GraWoLF, a scalable multiagent learning algorithm. The algorithm combines policy gradient ascent with the WoLF variable learning rate and can be applied effectively to complex tasks when agents have a variety of limitations. Its effectiveness was demonstrated on two very different multiagent learning tasks: goofspiel and keepout. Goofspiel is a card game with an intractably large state space, and keepout is an adversarial robot task. We also examined the problem of evaluating simultaneous multiagent learning algorithms in complex problems where limitations may eliminate the game's equilibria. We proposed three techniques for evaluating learning and applied these techniques to evaluating GraWoLF in our two tasks. Using these evaluation methods, we demonstrated that GraWoLF can effectively learn in the presence of other learning agents, even in complex tasks and faced with a variety of limitations.

Chapter 7

Team Adaptation

In this chapter we examine the problem of a team of agents learning when faced with an unknown opponent in a complex and dynamic environment. Specifically, we examine this problem within the context of autonomous robot soccer. The domain we explore is the Small-Size League of the RoboCup Initiative (Kitano, Kuniyoshi, Noda, Asada, Matsubara, & Osawa, 1997), where two teams of five robots compete against each other. This problem has many similarities to the domains that have been investigated in earlier chapters, but there are also some differences.

Specifically, RoboCup is not currently a domain of simultaneous learning. The team we describe in this chapter is, to our knowledge, the first RoboCup robot team to adapt to its current opponent during the course of a game. This fact simplifies many of the issues faced in the previous domains we examined, since we do not need to worry about convergence or equilibria. As such, we do not currently employ variable learning rates or the WoLF principle, since it is unnecessary when the other agents play a stationary strategy. On the other hand, this domain is considerably more complex, and the range of possible team play is extremely broad.

We introduce the concept of *plays* as a way of codifying team strategy and coordination. Plays then form the basis for injecting team adaptation. We give an overview of the CMDragons Small-Size robot soccer team, focusing on the role of plays within our control architecture. We then describe how we use a *playbook* to provide multiple team behavior options, which define the alternatives for our adapting team. We discuss the use of plays and playbooks in the RoboCup 2002 competition, as well as further evaluation within our robot simulator.

7.1 CMDragons'02

CMDragons'02 is a robot soccer team in the RoboCup Small-Size League. It is a fully autonomous team of robots with shared perception and off-board processing. We give an overview of the RoboCup Small-Size League emphasizing the challenges faced by an autonomous robot team. We then describe the CMDragons'02 system, focusing on how plays and play adaptation fit into the overall team architecture.

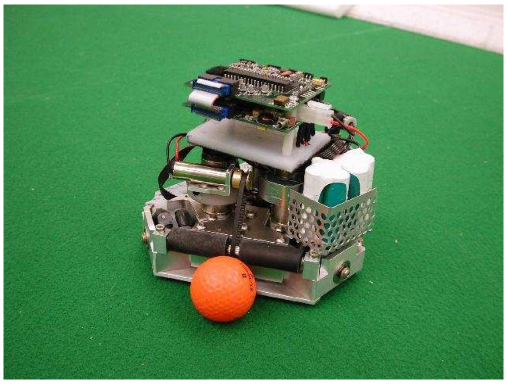
Figure 7.1: RoboCup small-size league field and robots preparing for a competition.

7.1.1 RoboCup Small-Size

RoboCup small-size robot soccer is a competition between two teams of five robots. The competition is played on a 2.8m by 2.3m field with short, sloped walls, an orange golf ball , and FIFA-like rules enforced by a human referee. Figure 7.1 shows a picture of the field and robots in preparation for a competition. In addition to local sensors, a team can mount a camera above the field to provide a global and shared perception of the field. Teams also make use of off-field computers as computation resources both to process visual information and to select team and individual actions. There is no standard robot platform, only restrictions on the size, shape, and allowed manipulators of the ball (e.g., non-goalie robots are not allowed to "hold" the ball). The CMDragons'02 team is heterogeneous, consisting of two types of robots. Figure 7.2(a) shows our differential-drive robots, which employ a two-wheel drive mechanism much like a wheelchair. Figure 7.2(b) shows our omni-directional robots, which employ a three-wheel mechanism to independently control direction and orientation. You can also observe the kicker and dribbling mechanism, which uses back spin to keep possession of the ball, on these robots.

Competitions are highly dynamic, where robots reach speeds over 1m/s and the ball often travels over 2m/s and as high as 5m/s. These speeds mean, with a standard camera capturing thirty frames per second, the system would have eight camera frames to respond to a high-powered shot from midfield. The inherent latency in the system often accounts for half of this time, leaving little time for deliberation and response. The constantly changing world, combined with the adversarial nature of a competing team with unknown capabilities and behavior, are the key challenges for a

(a) Diff-bots



(b) Omni-bots

Figure 7.2: CMDragons'02 robots. (a) shows our differential drive robots, and (b) shows our omni-directional robots.

robot team to overcome.

Since each team consists of five robots, it is also critical for the individual robots to act in a coordinated manner. Coordination requires both robust and predictable individual robot behaviors, as well as mechanisms for robots to combine their individual skills into an effective team behavior. The highly dynamic environment requires a reactive system to take advantage of short-lived opportunities. The competitive nature, though, requires longer-term deliberative actions to create these opportunities, e.g., passing, screens, deflections, and positioning for rebounds. Reaction and deliberation need to be balanced in an effective team.

Latency

Latency is another important factor in RoboCup Small-Size robot competitions. The complete control loop from receiving information through the camera, processing the information, deliberation, sending commands to the robots themselves, and execution combines to equal the system latency. Latency is the minimum amount of time between an event occurring in the environment and a change in the robot's behavior in response to that event. Equivalently, it is the amount of time between a robot choosing to take an action before the consequences of the action can be observed in the world.

Latency is both something that is very important to minimize, as well as something that is very important to account for. For a team's own robots, the effects of latency can be lessened by incorporating past commands into the state of the world. All deliberation then occurs based on the estimated state of the world when the command will be received, rather than the state of the world at the last observation. With accurate robot control models, most latency effects can be overcome. On the other hand, latency as it relates to the opponent robots does not have an easy solution. Although one can predict accurately the future state of one's own team, one cannot do the same for the opponent since their latent commands are not known. The result is that the true position and velocities of the opponent robots cannot be fully known. In addition, a team can take action knowing that the other team will be delayed in responding to its action.

Formal Frameworks

Throughout this work we have focused on the framework of stochastic games. The Small-Size League is a very complex domain, and could be thought of as a continuous state, partially observable, zero-sum stochastic game. The physical positions, orientations, velocities, and higher-order state of the robots and ball, along with the score, time remaining, and current game situation forms the state of the world. For a given robot, transitions of that robot's portion of the state are simply the actual stochastic outcome of these actions on their state. System latency adds further issues by requiring the state to include unobservable (by the opponent team) information about the latent robot commands. Partial observability is also present in that the systems all have sensor noise in perception, as well as frequent ball occlusion.

The largest difficulty in constructing such a model is that the actions and capabilities of opponent teams' robots are completely unknown. Some robots are omni-directional (e.g., Figure 7.2(b)) allowing them to drive in any direction while simultaneously changing their orientation. Other robots use a differential-drive system (e.g., Figure 7.2(a)), which prevents the robot from sideways movement. All robots have acceleration constraints, but this can vary between robots. Some robots have special ball-handling devices, such as kicking and dribbling mechanisms, which add to the action spaces of the robots. A theory of agent limitations could be used to capture this model. The action space available to the robots could be defined as a superset of all possible physically plausible actions allowed by the rules. Teams of robots by their design decisions impose limitations on this superset of actions making only a small set of these actions possible for any given team.

The exercise of applying a theory of agent limitations, though, is purely academic. Such a model is far beyond any practical use. It does, though, show that an equilibrium policy for a team in this setting may not exist. The uncertainty, which is due to an unknown adversary, and partial observability, which is due to latency and incomplete and noisy perception, place major restrictions on possible behavior. As we discussed in Chapter 5, we have no guarantee that an equilibrium exists with these limitations.

The behavior of most teams in the RoboCup Small-Size League, though, is to a large degree fixed. The behaviors are mostly reactive considerations based on the current estimate of the world state. Teams may enforce some commitment to assumptions, such as the position of the ball while it is occluded. Teams may also commit to a plan of attack, such as the same robot pursuing the ball over a period of time. Beyond these limited commitments, the overall team behavior does not change throughout the game, and especially does not change as a function of the opponent's behavior. Therefore, we can assume that the opponent has a fully limited restricted policy space, and the model becomes a POMDP, with a well-defined notion of an optimal policy. Even with this simplification, it is still not very practical since the opponent's behavior, although fixed, is unknown.

Although the models are impractical, they do help one to understand the goal for a Small-Size RoboCup team. Since the opponent is unknown, the goal is to play well against the range of likely opponents, with the possibility of adapting behavior during the course of the game to the specific current opponent. This is the basic goal for our CMDragons'02 team.

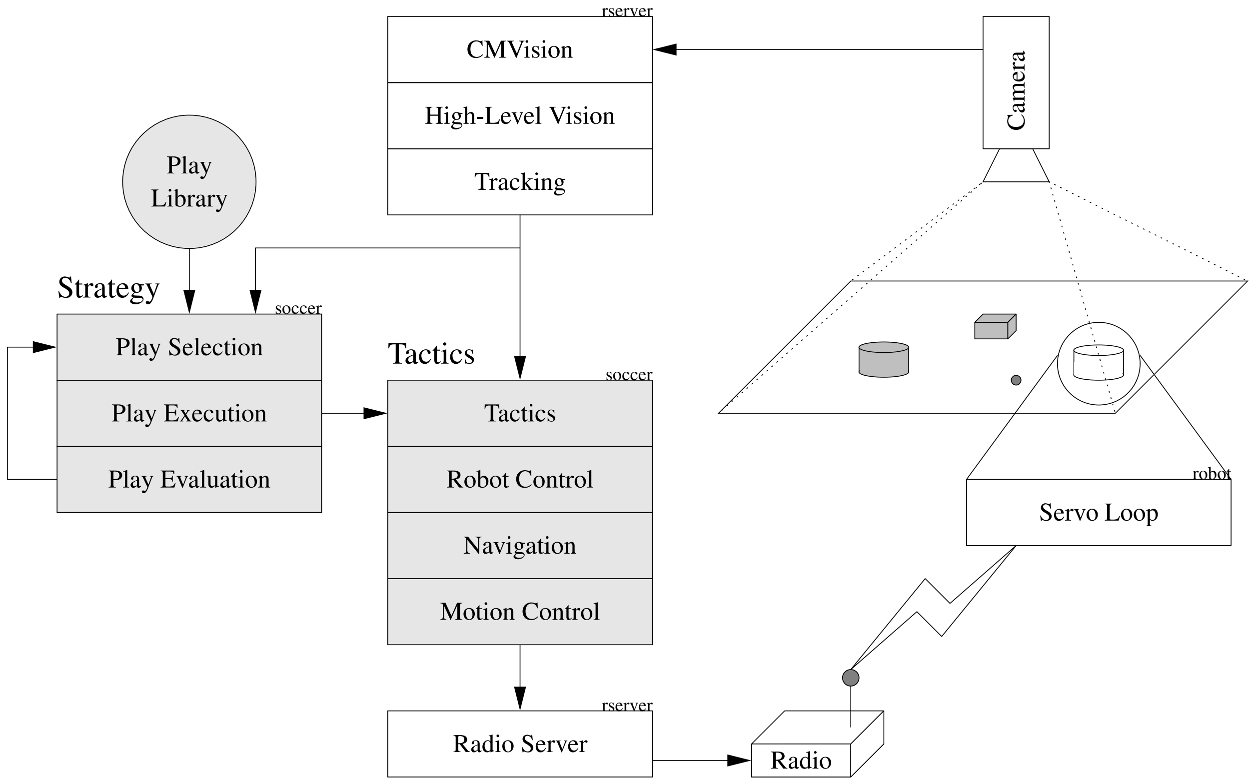


Figure 7.3: Overview of the CMDragons'02 team architecture.

7.1.2 System Architecture

An overview of the complete CMDragons'02 team architecture is shown in Figure 7.3. The perception-reasoning-action loop consists of taking a visual frame from the camera, processing the frame, determining controls for each robot on the team, and sending these controls to the robots using radio communication. The robots themselves perform a very high frequency servo loop using local encoder sensors to perform the commanded control. The vision processing uses team and robot markers to extract noisy estimates of the position and orientation of the team's robots, and the position of the opponent robots and the ball. The processed observation is then integrated into the team's shared world model using extended Kalman-Bucy filters, with a detailed model of the field physics and sensor noise (Browning, Bowling, & Veloso, 2002).

We focus on the control process, which is represented by the two shaded boxes in Figure 7.3. Our control system is broken into two layers: tactics and strategy. The tactics layer encompasses the individual robot behaviors, and each robot executes tactics independently of the other robots. The strategy layer provides the coordination mechanism and is executed for the entire team. This distinction allows us to separately focus on robust individual robot skills as well as powerful and adaptable team behavior. Plays comprise the core of the team's strategy layer, and are described in the next section. We first, though, consider the tactical level, which forms the building blocks for plays.

Tactics are defined to be any behavior executable by a single robot. "Shoot" is a prime example of a tactic. A list of tactics with brief descriptions is given in Table 7.1. Tactics are specified for each robot by the strategic play system and are executed independently. Tactics can actually be thought of as subgoals achievable by individual robots. The strategy layer's task is to find an appropriate set of tactics to achieve the overall team goal. The tactic layer's task is for a robot to achieve the tactic subgoal as quickly and as accurately as possible. Tactics are highly parameterized and allow for a very flexible range of definable behavior. We look at this further when we discuss how tactics are combined into plays to define a team behavior.

Tactics make use of a robot control layer that keeps robot specific information that persists even when a robot's tactic changes. This layer turns the tactic into waypoints used by the navigation layer. Fast navigation in an environment as dynamic as the RoboCup Small-Size competitions is a very challenging problem on its own. We use an extension of Rapidly-exploring Random Trees (RRTs), to provide fast, near-optimal, path planning and replanning for real-time multiple robot domains. Further details of the path planning algorithm are outside of the scope of this work (Bruce & Veloso, 2002).

The final layer of control is to compute appropriate robot velocities to move the robot along the path generated by the navigation algorithm. We use a trapezoidal velocity profile which models acceleration bounded constraints for robots. We decompose the problem into separate dimensions solving the one dimensional problem using the trapezoid profile and then recombining to get a target robot velocity. The computed robot velocities are then sent to the robot over radio communication. We now examine how to combine individual robot behaviors into coordinated team behavior. This is done through our play-based strategy engine.

In summary, tactics provide a layer of abstraction between the strategic and tactical behavior layers. For the purpose of the strategy layer, tactics can be thought of as specifying a policy, in the sense of selecting actions based on perceptions, for an individual robot. In this way, a set of tactics for the entire team is essentially a joint policy. The strategic layer, since it is responsible for assigning tactics, effectively specifies a joint policy.

7.2 Plays

The focus in this chapter is on the strategy component of our system. Given a set of tactics, which are effective and parameterized individual robot behaviors, how do we select each robot's behavior to achieve the team's goals? Our team strategy is built around the concept of a *play* as a team plan, and the concept of a *playbook* as a collection of plays. We first explore the goals for the design of a team's strategy and then explore how plays and playbooks achieve these goals. We later focus on how plays allow for team level adaptation to deal with an unknown opponent.

7.2.1 Goals

Obviously the main criterion for a team strategy system is performance. A single, monolithic team strategy that maximizes performance, though, is impractical. As we have seen repeatedly in this work, in an adversarial domain with an unknown opponent, a single optimal strategy is unlikely to exist. Even concepts of equilibria may not exist when dealing with systems with physical and rational limitations. Instead we enumerate a set of six simpler goals, which we believe are achievable and lead to strong overall team performance:

Tactic	Description
shoot	Manipulate the ball to go into the goal.
steal	Manipulate the ball to remove possession of it from another robot.
clear	Manipulate the ball to go down field, away from one's own goal.
active_def	Switches between steal and clear as appropriate.
pass	Manipulate the ball toward a teammate.
dribble_to_position	Manipulate the ball to a place on the field while keeping possession.
dribble_to_region	Manipulate the ball to a region of the field while keeping possession.
dribble_to_shoot	Manipulate the ball to a region of the field where a good shot can be taken while keeping possession.
spin_to_region	Using a spinning maneuver to move the ball to a region on the field.
receive_pass	Move to receive a pass enroute.
receive_deflection	Move to deflect a pass toward the goal.
position_for_start position_for_kick position_for_penalty charge_ball	Special tactics used for set-plays such as kickoffs, free kicks, and penalty shots.
position_for_loose_ball position_for_rebound position_for_pass position_for_deflection	Move to a position on the field to anticipate a loose ball, rebound, pass, or deflection.
defend_line	Defend the ball from crossing a particular line on the field.
defend_point	Defend the ball from a particular point on the field.
defend_lane	Defend along a particular lane.
block	Defend the goal a particular distance from the goal.
mark	Mark an opponent preventing them from getting the ball, getting to the goal, blocking a shot.
goalie	A special tactic just for the goalie.
stop velocity position	Basic tactics specifying a particular velocity or position for the robot.

Table 7.1: List of tactics along with a brief description.

1. Coordinated team behavior,

2. Temporally extended sequences of action (deliberative),

3. Inclusion of special purpose behavior for certain circumstances,

4. Ease of human design and augmentation,

5. Ability to exploit short-lived opportunities (reactive), and

6. On-line adaptation to the specific opponent,

The first four goals require plays to be able to express complex, coordinated, and sequenced behavior among teammates. In addition, the language must be human readable to make play design and modification simple. These goals also require a powerful system capable of executing the complex behaviors the plays describe. The fifth goal requires the execution system to also recognize and exploit opportunities that are not explicitly described by the current play. Finally, the sixth goal requires the system to alter its overall behavior over time. Notice that the strategy system requires both deliberative and reactive reasoning. The dynamic environment makes a strictly deliberative system unlikely to be able to carry out its plan, but the competitive nature often requires explicitly deliberative sequences of actions in order to create scoring opportunities.

We first introduce our novel play language along with the coupled play execution system. We later examine how a playbook can be used to adapt to an opponent.

7.2.2 Play Specification

A play is a multiagent plan, i.e., a joint policy for the entire team. Our definition of a play, therefore, shares many concepts with classical planning. A play consists of four main components:

- Applicability conditions,

- Termination conditions,

- Roles, and

- Execution details.

Applicability conditions specify when a play can be executed and are similar to planning operator preconditions. Termination conditions define when execution is stopped and are similar to an operator's effects, although they include a number of possible outcomes of execution. The roles describe the actual behavior to be executed in terms of individual robot tactics. The execution details can include a variety of optional information that can help guide the play execution system. We now look at each of these components individually.

```
PLAY Naive Offense

APPLICABLE offense
DONE aborted !offense

ROLE 1
  shoot A
  none
ROLE 2
  defend_point {-1400 250} 0 700
  none
ROLE 3
  defend_lane {B 0 -200} {B 1175 -200}
  none
ROLE 4
  defend_point {-1400 -250} 0 1400
  none
```

Table 7.2: A simple example of a play.

Applicability Conditions

The conditions for a play's applicability can be defined as any logical formula of the available state predicates. The conditions are specified as a logical DNF using the `APPLICABLE` keyword, with each disjunct specified separately. In the example play in Table 7.2, the play can only be executed from a state where the `offense` predicate is true. The `offense` predicate is actually a fairly complex combination of the present and past possession of the ball and its present and past position on the field. Predicates can be easily added and Table 7.3 lists the current predicates used by our system. Note that predicates can also take parameters, as in the case of `ball_x_gt` X, which checks if the ball is over the distance X down field.

Like preconditions in classical planning, applicability conditions restrict when a play can be executed. By constraining the applicability of a play, one can design special purpose plays for very specific circumstances. An example of such a play is shown in Table 7.4. This play uses the `ball_in_their_corner` predicate to constrain the play to be executed only when the ball is in a corner near the opponent's goal. The play explicitly involves dribbling the ball out of the corner to get a better angle for a shot on goal. Such a play only really makes sense when initiated from the play's applicability conditions.

Termination Conditions

Termination conditions specify when the play's execution should stop. Just as applicability conditions are related to operator preconditions in classical planning, termination conditions are similar to operator effects. Unlike classical planning, though, there is too much uncertainty in execution

offense	our_kickoff
defense	their_kickoff
their_ball	our_freekick
our_ball	their_freekick
loose_ball	our_penalty
ball_their_side	their_penalty
ball_our_side	ball_x_gt *X*
ball_midfield	ball_x_lt *Y*
ball_in_our_corner	ball_absy_gt *Y*
ball_in_their_corner	ball_absy_lt *Y*
nopponents_our_side *N*	

Table 7.3: List of state predicates.

```
PLAY Two Attackers, Corner Dribble 1

APPLICABLE offense in_their_corner
DONE aborted !offense
TIMEOUT 15

ROLE 1
        dribble_to_shoot { R { B 1100 800 } { B 700 800 } 300}
        shoot A
        none
ROLE 2
        block 320 900 -1
        none
ROLE 3
        position_for_pass { R { B 1000 0 } { B 700 0 } 500 }
        none
ROLE 4
        defend_line { -1400 1150 } { -1400 -1150 } 1100 1400
        none
```

Table 7.4: A special purpose play that is only executed when the ball is in an offensive corner of the field.

to know the exact outcome of a particular play. The termination conditions list possible outcomes and associate a *result* with each possible outcome. The soccer domain itself defines a number of stopping conditions, e.g., the scoring of a goal or the awarding of a penalty shot. The play's termination conditions are in addition to these and allow for play execution to be stopped and a new play initiated even when the game itself is not stopped.

Termination conditions, like applicability conditions, use logical formulas of state predicates. In addition to specifying a conjunction of predicates, a termination condition also specifies the result of the play if the condition becomes true. In the play specification, they are delineated by the `DONE` keyword, followed by the result, and then the list of conjunctive predicates. Multiple `DONE` conditions can be specified and are interpreted in a disjunctive fashion. In the example play in Table 7.2, the only terminating condition, beside the default soccer conditions, is if the team is no longer on offense ("!" is used to signify negation). The play's result is then "aborted".

The results for plays are one of: succeeded, completed, aborted, and failed. These results are used to evaluate the success of the play for the purposes of reselecting the play later. This is the major input to the team adaptation system, which we describe later. Roughly speaking, we use results of succeeded and failed to mean that a goal was scored, or some other equivalently valuable result, such as a penalty shot. the completed result is used if the play was executed to completion. For example, in the play in Table 7.2, if a robot was able to complete a shot, even if no goal was scored, the play is considered completed. In a defensive play, switching to offense may be a completed result in the `DONE` conditions. The aborted result is used when the play was stopped without completing.

Besides `DONE` conditions, there are two other ways in which plays can be terminated. The first is when the sequence of behaviors defined by the play are executed. As we mentioned above, this gives the play a result of `completed`. This will be described further when we examine the play execution system. The second occurs when a play runs for a long time with no other termination condition being triggered. When this occurs the play is terminated with an `aborted` result and a new play is selected. This allows the team to commit to a course of action for a period of time, but recognize that in certain circumstances a particular play may not be able to progress any further.

Roles

As plays are multiagent plans, the main component are the roles. Each play has four roles, one for each non-goalie robot on the field. A role consists of a list of behaviors for the robot to perform in sequence. In the example play in Table 7.2, there is only a single behavior listed for each role. These behaviors will simply be executed until one of the termination conditions apply. In the example play in Table 7.4, the first role has two sequenced behaviors. In this case the robot will dribble the ball out of the corner. After the first tactic finishes, the robot filling that role will switch to the `shoot` tactic and try to manipulate the ball toward the goal.

Sequencing also requires coordination, which is a critical aspect of multiagent plans. Coordination in plays requires all the roles to transition simultaneously through their sequence of behaviors. For example, consider the more complex play in Table 7.5. In this play, one player is assigned to pass the ball to another player. Once the pass behavior is completed *all* the roles transition to their next behavior, if one is defined. So, the passing player will switch to a mark behavior, and the target of the pass will switch to a behavior to receive the pass, after which it will switch to a

```
PLAY Two Attackers, Pass

APPLICABLE offense
DONE aborted !offense

OROLE 0 closest_to_ball

ROLE 1
  pass 3
  mark 0 from_shot
  none
ROLE 2
  block 320 900 -1
  none
ROLE 3
  position_for_pass { R { 1000 0 } { 700 0 } 500 }
  receive_pass
  shoot A
  none
ROLE 4
  defend_line { -1400 1150} {-1400 -1150} 1000 1400
  none
```

Table 7.5: A complex play involving sequencing of behaviors.

shooting behavior.

Roles are not tied to any particular robot. Instead, they rely on the play execution system to do this role assignment. The order of the roles presented in the play act as hints to the execution system for filling the roles. Roles are always listed in order of priority. The first role is always the most important and usually involves some manipulation of the ball. This provides the execution system the knowledge needed to select robots to perform the roles and also for role switching when appropriate opportunities present themselves.

Tactics in Roles. The different behaviors that can be specified by a role are the individual robot tactics that were discussed in Section 7.1.2. As mentioned, these tactics are highly parameterized behaviors. For example, the `defend_point` tactic takes a point on the field and a minimum and maximum range. The tactic will then position itself between the point and the ball, within the specified range. By allowing for this large degree of parameterization the different behaviors can be combined into a nearly infinite number of play possibilities. The list of parameters accepted by the different tactics is shown in Table 7.6.

Active Tactics

shoot (Aim | Noaim | Deflect ⟨*role*⟩)
steal [⟨*coordinate*⟩]
clear
active_def [⟨*coordinate*⟩]
pass ⟨*role*⟩
dribble_to_shoot ⟨*region*⟩
dribble_to_region ⟨*region*⟩
spin_to_region ⟨*region*⟩
receive_pass
receive_deflection
dribble_to_position ⟨*coordinate*⟩ ⟨*theta*⟩
position_for_start ⟨*coordinate*⟩ ⟨*theta*⟩
position_for_kick
position_for_penalty
charge_ball

Non-Active Tactics

position_for_loose_ball ⟨*region*⟩
position_for_rebound ⟨*region*⟩
position_for_pass ⟨*region*⟩
position_for_deflection ⟨*region*⟩
defend_line ⟨*coordinate-1*⟩ ⟨*coordinate-2*⟩ ⟨*min-dist*⟩ ⟨*max-dist*⟩
defend_point ⟨*coordinate-1*⟩ ⟨*min-dist*⟩ ⟨*max-dist*⟩
defend_lane ⟨*coordinate-1*⟩ ⟨*coordinate-2*⟩
block ⟨*min-dist*⟩ ⟨*max-dist*⟩ ⟨*side-pref*⟩
mark ⟨*orole*⟩ (ball | our_goal | their_goal | shot)
goalie
stop
velocity ⟨*vx*⟩ ⟨*vy*⟩ ⟨*vtheta*⟩
position ⟨*coordinate*⟩ ⟨*theta*⟩

Table 7.6: List of tactics with their accepted parameters.

Coordinate Systems. Many of the tactics take parameters in the form of "coordinates" or "regions". These parameters can be specified in a variety of coordinate systems allowing for added flexibility in specifying plays in general terms. We allow coordinates to be specified either as absolute field position or ball relative field positions. In addition, the positive y-axis can also be specified to depend on the side of the field that the ball is on, the side of field that the majority of the opponents are on, or even a combination of these two factors. This allows tremendous flexibility in the specification of the behaviors used in plays. Regions use coordinates to specify non-axis aligned rectangles as well as circles. This allows, for example, a single play to be general with respect to the side of the field and position of the ball.

Execution Details

The rest of the play specification are execution details, which amount to providing hints to the execution system about how to execute the play. These optional components are: timeout and opponent roles. The timeout overrides the default amount of time a play is allowed to execute before aborting the play and selecting a new play.

Opponent roles allow robot behaviors to refer to opponent robots in defining their behavior. The play in Table 7.5 is an example of this. The first role, switches to marking one of the opponents after it completes the pass. The exact opponent that is marked depends upon which opponent was assigned to opponent Role 0. Before the teammate roles are listed, opponent roles are defined by simply specifying a selection criteria for filling the role. The example play uses the `closest_to_ball` criterion, which assigns the opponent closest to the ball to fill that role, and consequently be marked following the pass. Multiple opponent roles can be specified and they are filled in turn using the provided criterion.

7.2.3 Play Execution

The play execution module is responsible for actually instantiating the play into real robot behavior. That is, the module must interpret a play by assigning tactics to actual robots. This instantiation consists of key decisions: role assignment, role switching, sequencing tactics, opportunistic behavior, and termination.

Role assignment uses tactic-specific methods for selecting a robot to fill each role, in the order of the role's priority. The first role considers all four field robots as candidates to fill the role. The remaining robots are considered to fill the second role, and so on. Role switching is a very effective technique for exploiting changes in the environment that alter the effectiveness of robots fulfilling roles. The play executor handles role switching using the tactic-specific methods for selecting robots, using a bias toward the current robot filling the role. Sequencing is needed to move the entire team through the sequence of tactics that make up the play. The play executor monitors the current *active player*, i.e., the robot whose role specifies a tactic related to the ball (see Table 7.1). When the tactic succeeds, the play is transitioned to the next tactic in the sequence of tactics, for *each* role. Finally, opportunistic behavior accounts for changes in the environment where a very basic action would have a valuable outcome. For example, the play executor evaluates the duration of time and potential success of each robot shooting immediately. If an opportunistic behavior can be executed quickly enough and with a high likelihood of success, then the robot immediately switches its behavior to take advantage of the situation. If the opportunity is then lost, the robot returns to executing its role in the play.

The play executor algorithm provides basic behavior beyond what the play specifies. The play executor, therefore, simplifies the creation of plays, since this basic behavior does not need to be considered in the design of plays. The executor also gives the team robustness to a changing environment, which can cause a play's complex behavior to be no longer necessary or require some adjustment to the role assignment. It also allows for fairly complex and chained behavior to be specified in a play, without fear that short-lived opportunities will be missed.

The final consideration of play execution is termination. We have already described how plays specify their own termination criteria, either through predicates or a timeout. The executor checks these conditions, and also checks whether the play has completed its sequence of behaviors, as well as checking incoming information from the referee. If the final active tactic in the play's sequence of tactics completes, then the play is considered to have completed and is terminated. Alternatively, the game may be stopped by the referee to declare a penalty, award a free kick, award a penalty kick, declare a score, and so on. Each of these conditions terminates the play, but also may effect the determined outcome of the play. Goals are always considered successes or failures, as appropriate. Penalty kicks are also considered play successes and failures. A free kick for our team deems the play as completed, while a free kick for the opponent sets the play outcome to aborted. Play outcomes are the critical input to the play selection and adaptation system.

7.2.4 Playbook and Play Selection

Plays define a team plan. A playbook is a collection of plays, and, therefore, provides a whole range of possible team behavior. Playbooks can be composed in a number of different fashions. For example, one could insure that for all possible game states there exists a single applicable play. This makes play selection simple since it merely requires executing the one applicable play from the playbook. A more interesting approach is to provide multiple applicable plays for various game states. This adds a play selection problem, but also adds alternative modes of play that may be more appropriate for different opponents. Multiple plays also give options from among which adaptation can select. In order to support multiple applicable plays, a playbook also associates a weight with each play. This weight corresponds to how often the play should be selected when applicable.

Play selection, the final component of the strategy layer, then amounts to finding the set of applicable plays and selecting one based on the weights. Specifically, if $p_{1...k}$ are the set of plays whose applicability condition are satisfied, and w_i is their associated weight, then p_j is selected with probability,

$$Pr(p_j|\mathbf{w}) = \frac{w_j}{\sum_{i=1}^{k} p_i}.$$

Although these weights can simply be specified in the playbook and left alone, they also are the parameters that can be adapted for a particular opponent. In Section 7.3 we describe how these weights can be changed based on past play outcomes. We will also show experiments evaluating the importance of adaptation when the domain has a great degree of uncertainty. In our case, the uncertainty stems from the unknown opponents.

7.2.5 Achieving Our Goals

Our play-based strategy system, thus far described, achieves the first five goals that we set out in Section 7.2.1. Sequences of synchronized actions provide a mechanism for coordinated team behavior, as well as deliberative actions. Applicability conditions allow for the definition of special purpose team behavior. The play execution system handles moments of opportunity to allow for the team to have a reactive element. Finally, incorporating all of this into a human readable text format makes adding and modifying plays quite easy. The final goal, which has not been addressed, is adapting team behavior. We now examine how to incorporate adaptation into the play selection process and evaluate its effectiveness.

7.3 Team Adaptation

Team adaptation is the problem of modifying the selection of plays from the playbook based on past execution. The basic premise is that the best play to select depends upon the opponent, which is unknown. Our technique has a strong similarity to WoLF and GraWoLF presented in earlier chapters. We base adaptation on the outcome of the team's own actions, rather than on examining the specific behavior of the opponent. The adaptation directly address the goal, which is to better select actions over time.

The team's actions are plays, and so the outcome of the team's actions are the outcomes of the plays. We use the play evaluation results (success, failure, completed, and aborted) of past selected plays to compute new weights for selecting the next play. A play adaptation rule, W, takes a weight vector, $\mathbf{w}$, the executed play, p, and the outcome of the play, o, and returns the play's new weight, i.e.,

$$W(\mathbf{w}, p, o) \rightarrow (0, \infty)$$

There are a number of desirable properties of this rule. For example, an outcome of success or complete should not decrease the play's weight. Similarly, aborts and failures should not increase the weight. Another important feature is that any outcome should have a substantial effect on the play's weight. This is necessary since the short time span of a single game requires dramatic weight changes if adaptation is to have an effect on the game's outcome.

7.3.1 Competition

Our strategy system was used effectively during RoboCup 2002 against a wide range of opponents with vastly different strategies and capabilities. We implemented a simple adaptation rule using weight multiplication, where each outcome multiplies the play's previous weight by a constant. Specifically, the rule is,

$$W(\mathbf{w}, p, o) = \begin{cases} w_p S & \text{if } o = \texttt{succeeded} \\ w_p C & \text{if } o = \texttt{completed} \\ w_p A & \text{if } o = \texttt{aborted} \\ w_p F & \text{if } o = \texttt{failed} \end{cases},$$

where S, C, A, F are the multiplication factors. For our specific implementation these values were set to,

$$S = 4 \quad C = \frac{4}{3} \quad A = \frac{3}{4} \quad F = \frac{1}{4}.$$

These weights are extremely aggressive. For example, scoring a goal would effective quadruple the likelihood of selecting the play again. Likewise, being scored against would make it four times less likely to be selected. Since success and failure (as well as complete and abort) are roughly equivalent and opposite outcomes, we set the multipliers to be inverses. Our weight of completions to successes was selected arbitrarily to give a significant weight to goals, while still allowing other outcomes to effect the selection.

The competition results are difficult to evaluate in a systematic way. We present here some anecdotal evaluation based on the competition. Overall the CMDragons team performed well in the competition. The team advanced out of the preliminary round with a record of three wins and one loss, and was eliminated in the quarter finals.

Two observations have resulted from the competition experience, both related to the game against the previous year's champion team, Lucky Star, from Ngee Ann Polytechnic University in Singapore. This was our only loss in the preliminary round and we were defeated 2-0. The first interesting observation is that both goals occurred in the first half, allowing no goals in the second half, which suggests that our team effectively adapted to the opponent's style of attack. At the end of the game, the weights were strongly in favor of one particular play, which positioned a defender to block off half the field to prevent crosses or rebounds from getting to the other side of the field. Whether this result is a product of adaptation or just a statistical anomaly cannot be known for certain, which is the problem with competition results. The second observation is that due to Lucky Star's speed and maneuverability, none of the offensive plays could score with any regularity. Adaptation continued trying a variety of plays as abort results decreased the executed play's weight. However, adaptation among a set of plays cannot do any better than the best play in the playbook. Although, not a surprising fact, it was illustrated to our disadvantage during this game.

7.3.2 Evaluation

The nature of robot competitions prevent them from being a systematic evaluation in a controlled or statistically significant way. Therefore, we have constructed a number of simplified scenarios to evaluate our play-based team strategy. These scenarios compare whether multiple plays are actually necessary, and also examine the usefulness of playbook adaptation.

The scenarios compare the effectiveness of four different offensive plays paired against three different defensive behaviors. To simplify evaluation, only two offensive robots were used against one defensive robot. The robots start in the usual "kick off" position in the center of the field. For each scenario, 750 trials were performed in our UberSim Small-Size League simulator (Browning & Tryzelaar, 2003). A trial was considered a success if the offense scored a goal within a 20 second time limit, and a failure otherwise. Table 7.7 lists the specifics of the offensive plays and defensive behaviors. The first two defensive behaviors, and first three offensive plays, formed the core of the playbook used for RoboCup 2002.

Figure 7.4 displays an example trace of one such trial. The trace is shown as a series of frames, *Offensive Plays*

Name	Attacker 1	Attacker 2
Shoot 1	`shoot Aim`	`position_for_rebound`
Shoot 2	`shoot NoAim`	`position_for_rebound`
Screen 1	`shoot Aim`	`mark 0 from_shot`
Screen 2	`shoot NoAim`	`mark 0 from_shot`

Defensive Behaviors

`block`	Positions to block incoming shots
`active_def`	Actively tries to steal ball
`brick`	Defender does not move

Table 7.7: List of offensive and defensive behaviors tested.

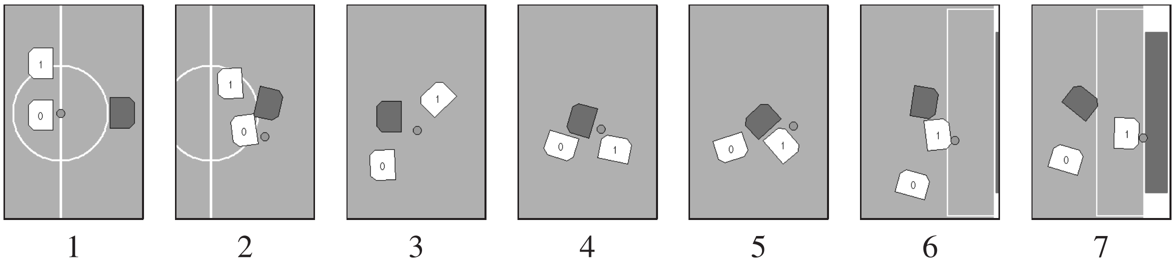


Figure 7.4: An example trace of play Screen 2 against defensive behavior `active_def`.

shown in temporal order (Frame 1 represents the start of the trial, with the robots in the "kick-off" position). In each frame, the two numbered robots are the offensive team. The remaining robot is on the defensive team, and the small shaded circle is the ball. In the first two frames, Robot 0, which is running the `shoot` tactic, attempts to drive around the defensive robot, but loses control of the ball due to a collision. Meanwhile, Robot 1, which is running the `mark` tactic, tries to impede the defensive robot (frames 3-5). This approach is successful in this case, and Robot 1 forces a turnover. Immediately afterward, robot 1 switches roles and becomes the shooting attacker in order to opportunistically score (frames 5-7). This example serves to illustrate one possible way robots may be coordinated using plays.

Table 7.8 shows the play comparison results. Each trial is independent, and so the maximum likelihood estimate of each play's success probability is the ratio of successes to trials. Note that there is no static strategy that is optimal against every possible opponent. Even in this simplified scenario, the best strategy depends upon the opposing defensive behavior. In fact, each of the offensive plays is actually the optimal response for some distribution over defensive behaviors.

Our results support the notion that play-based strategies are capable of exhibiting many different behaviors with varying degrees of effectiveness. For instance, the screen plays, one of which was shown in the example trace, are effective against an "active" defender that tries to steal the ball from the offense, because the non-shooting attacker is capable of setting screens for the shooting attacker. On the other hand, the screen plays are less effective against a "blocking" defender that guards the goal. Against such defenses, the Shoot 1 and 2 plays, where the non-shooting robot attempts to position itself for rebounds, are far more effective. Thus we use team adaptation to select the best offensive strategy against the particularly encountered defense.

Play	`block`	`active_def`	`brick`
Shoot 1	**72.3%**	49.7%	**99.5%**
Shoot 2	66.7%	57.3%	43.1%
Screen 1	40.8%	59.0%	92.4%
Screen 2	49.2%	**66.0%**	72.0%

Table 7.8: Play comparison results. For each scenario, the percentage of successes for the 750 trials is shown. The boldfaced number corresponds to the play with the highest percentage of success for each defensive behavior.

To explore the effectiveness of playbook adaptation, we experiment with an offensive playbook for two robots with all four offensive plays described in Table 7.7 against a fixed defender running either `block` or `active_def`. We initially used the algorithm we used in competition, but discovered an imperfection in the approach. Due to the strength of the reinforcement for a completed play, it is possible for a moderately successful but non-dominant play to quickly dominate in weight, and remain dominant. This phenomenon did not occur in competition due to the larger disparity in plays against a given opponent and lower success probabilities. The problem is that there is no normalization in the weight adjustment for plays that have a higher selection probability, and so are updated more often. Therefore, we included a normalization factor in the weight updates. Specifically, we used the following rule,

$$W(\mathbf{w}, p, o) = \begin{cases} 2w_p / Pr(p|\mathbf{w}) & \text{if } o = \texttt{Succeeded} \\ w_p \Pr(p|\mathbf{w})/2 & \text{if } o = \texttt{Failed} \end{cases},$$

where $\Pr(p|\mathbf{w})$ is the probability assigned to p according to $\mathbf{w}$.

To evaluate the performance of play adaptation using this rule, we compare the expected success rate (ESR) of using this adaptation rule against a fixed defensive behavior. We used the results in Table 7.8 to simulate the outcomes of the various play combinations. Figure 7.5(a) and (b) show the ESR for play adaptation over 100 trials, which is comparable to the length of a competition (approximately 20 minutes). The lower bound on the y-axis corresponds to the ESR of randomly selecting plays and the upper bound corresponds to the ESR of the playbook's best play for the particular defense. Figure 7.5(c) shows the probabilities of selecting each play over time when running the adaptation algorithm.

As Figure 7.5(a) and (b) indicates, against each defense the overall success rate of the offense quickly grows toward the optimal success rate within a small number of trials. Likewise Figure 7.5(c) shows that against the `block` defense, the probability of selecting either of two plays with comparatively high individual success rates quickly dominates the probability of selecting the two less successful plays. Clearly, the algorithm very quickly favors the more successful plays.

These results, combined with the RoboCup performances, demonstrate that adaptation can be a powerful tool for identifying successful plays against unknown opponents. Note the contrast here between the use of adaptation and more common machine learning approaches. We are not interested in convergence to an optimal control policy. Rather, given the short time limit of a game, we desire adaptation that achieves good results quickly enough to impact the game. Hence a fast, but non-optimal response is desired over a more optimal but longer term approach.

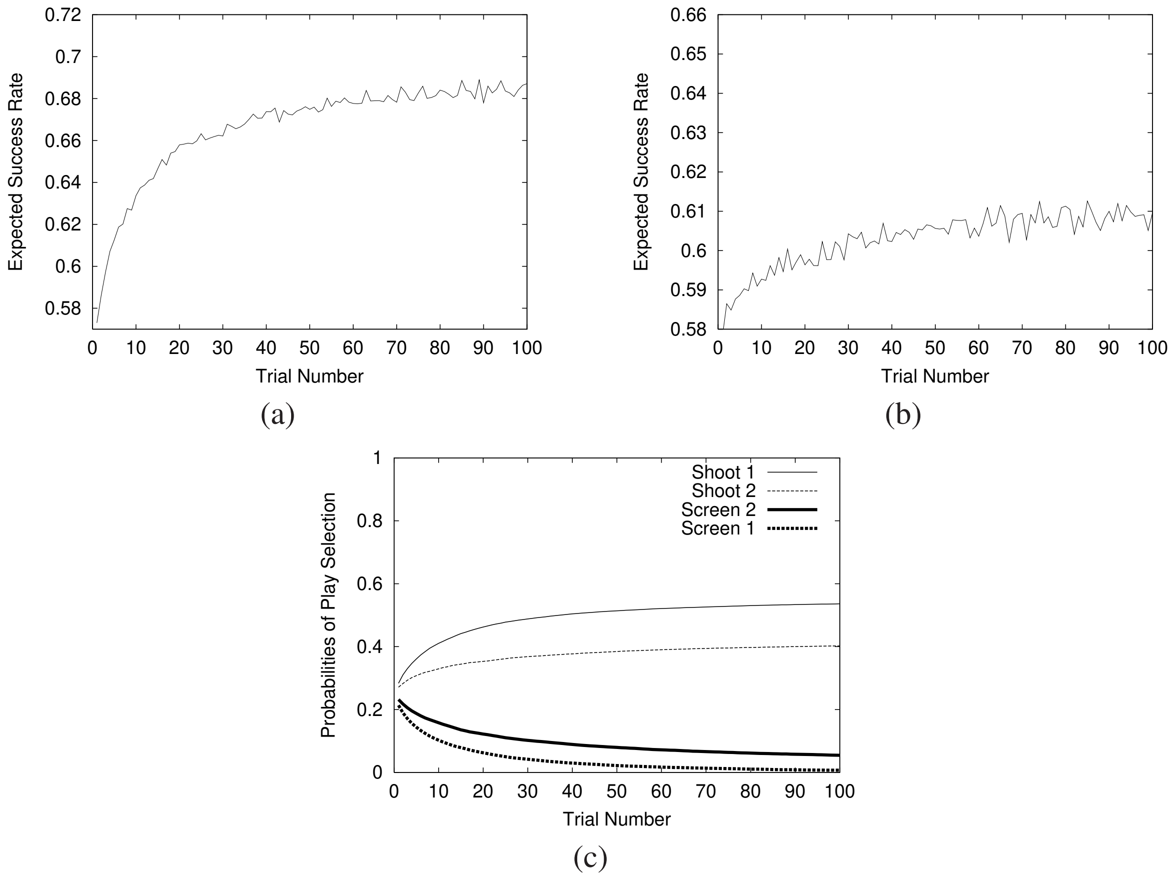


Figure 7.5: Expected success rates over time versus the defensive behavior (a) block and (b) active_def. (c) Expected play probabilities over time versus block. These results have all been averaged over 50000 runs of 100 trials.

7.3.3 Discussion

There are a number of qualitative advantages of using plays for team coordination: the relative simplicity and cleanliness of the approach, and the ease with which new strategies can be specified. Our decomposition of the team planning problem is highly hierarchical and modular. At the lowest level, tactics specify individual robot behaviors; in the middle level, these tactics are coordinated using the play language; and at the highest level, team adaptation specifies the overall behavior of the team. This particular way of structuring the team allows us to easily understand, test, and improve the system. Individual robot skills, written as tactics, end up being highly reusable and easily incorporated into multiple plays. Moreover, a new team behavior may take only minutes to write using the play language, after which it can be evaluated immediately.

The exact implementation of the team adaptation rule can still be improved upon. One possibility is to use a no-regret learning algorithm as discussed in Section 3.2.2. These algorithms would fit very nicely into this experimental setup, and additionally could, over time, guarantee performance that is nearly as good as the best play available in the playbook. On the other hand, it is not clear how well these techniques would fair in the short time space of an actual game. Also, if more teams begin employing adaptation during the course of a game, then the WoLF principle used in the previous chapters could be used to help handle simultaneous learning. As with no-regret algorithms, how well WoLF would perform in the short confines of the game is part of our future research plans.

As a whole, this system addresses all of our goals set out in Section 7.2.1 for our team strategy. It merges deliberation using coordinated extended sequences of actions, with opportunistic reaction. In addition, it provides the ability to define circumstance-specific special purpose plays, along with easy design and modification. Finally, it provides the mechanism for evaluating execution outcome and using this outcome to adapt play selection to the particular opponent.

7.4 Summary

In this chapter, we described a play-based team architecture for our CMDragons '02 team of robot soccer agents. We demonstrated how plays codify team behavior and allow for online evaluation of team performance for the particular opponent. This evaluation can then be used to select between play alternatives. The robot team can use this evaluation to autonomously alter its play selection, and therefore team behavior, to adapt to its particular opponent. This multiagent planning and adaptation system was used effectively in the RoboCup 2002 competition, and was the first RoboCup robot team to adapt to its opponent during the course of a game. We also performed evaluations of this system in simulation. We demonstrated that plays do have different results depending upon the opponents' behavior, and that adaptation can successfully improve team performance within the course of a game.

Chapter 8

Conclusions and Future Work

This dissertation set out to address the question,

> Can an agent effectively learn to act in the presence of other learning agents in complex domains when agents may have limitations?

In this final chapter we summarize the contributions we have presented to answer this question. We also describe some new directions for future work that these contributions raise.

8.1 Contributions

There were four main contributions of this thesis.

Variable learning rate and the WoLF principle for rational and convergent learning. In Chapter 4, we introduced two desirable properties for a multiagent learning algorithm and demonstrated that previous techniques do not simultaneously achieve these properties. We then presented two new concepts: a variable learning rate and the WoLF principle. We demonstrated these techniques to be a powerful and effective mechanism for making rational and convergent learners.

The WoLF variable learning rate has both theoretical justification in making the non-converging gradient ascent algorithm converge in a subclass of matrix games. We also demonstrated this converging effect empirically across a wide variety of multiagent domains: zero-sum, general-sum, two-player, three-player, matrix games, and stochastic games. The power of the WoLF variable learning rate is also further demonstrated in its effective use in GraWoLF, another contribution of this work and summarized below.

Analysis of restricted policy spaces and their effect on equilibria. The concept of equilibria is a critical foundation for most of the multiagent learning research to date. However, complete rationality in the sense of optimal action selection is not possible for realistic tasks. In fact, the research on scaling single-agent reinforcement learning to tasks with large and complex state spaces has explicit sacrificed optimality for speed of learning. Optimality, though, is the basis for equilibria, and so common and realistic limitations pose a substantial threat to this foundational concept. This thesis not only recognized and introduced the threat of limitations to equilibria, but also presented an analysis of their impact.

In Chapter 5 we contributed two models of the effect of limitations on agent behavior: implicit games and restricted policy spaces. We extended the concept of equilibria to these models and analyzed when equilibria are guaranteed to exist. In general, equilibria may not exist even with well-behaved agent limitations. We did find a number of broad classes of stochastic games and limitations for which equilibria can be guaranteed. This analysis has many implications for further research in multiagent learning on many different fronts.

GraWoLF as a general-purpose scalable multiagent learning algorithm. GraWoLF, presented in Chapter 6, combines the WoLF variable learning rate with a policy gradient learning technique to effectively learn in complex tasks. Learning is possible even when the learner and others are limited, which is unavoidable in tasks that are realistic and interesting. The algorithm was successfully applied to learning in two complex tasks: a card game with an intractably large state space, and an adversarial robot task. These results demonstrate effective learning in the presence of other learning agents, in complex domains, and in the presence of agents with a variety of limitations.

In addition to the algorithm and compelling results of its application in these two tasks, we also investigated the problem of evaluation. We proposed a number of techniques for evaluation of multiagent learning in situations of simultaneous learning with limited agents. We employed these three techniques, challengers, improvement over the initial policy, and expected value during learning, in evaluating our own results. These three techniques are generally useful tools for the difficult problem of evaluation of simultaneous learning.

A summary of the simultaneous learning domains that were investigated in this work is shown in Table 8.1. This table lists both the key details of the game as well as a summary of which algorithm was investigated in the domain, whether limited or unlimited agents were examined, and whether results were specifically in situations of self-play or went beyond self-play. Notice that the examined domains increase in complexity, beginning with simple two-player, two-action matrix games, where WoLF-IGA was examined theoretically, and ending with keepout, an adversarial, continuous-state, robot learning task, for which GraWoLF was applied.

Play-based team strategy for adapting to an unknown opponent. Our final contribution, presented in Chapter 7, is the development of a team strategy capable of learning and adapting to an unknown opponent. The mechanisms were designed and implemented in the CMDragons '02 RoboCup Small-Size autonomous robot team, but also have general applicability. Plays define team plans for achieving a task and are sufficiently descriptive to allow for sequences of behavior, special purpose activation, dynamic role assignment and role switching, and determination of the play's success or failure. These are all crucial elements for building an adaptable team. Learning is injected into the play selection algorithm through the use of our determination of play success and failure. Successful plays are weighted to be executed more often, while unsuccessful plays are executed less often. Using this team strategy, CMDragons'02 was the first RoboCup robot team to adapt to its opponent during the course of a game. We also presented the results of controlled experiments in a simulated environment to investigate the effectiveness of plays and play-based adaptation.

Domain	Matrix Game	Stochastic Game	Zero-Sum	Non-Zero-Sum	Number of Players	Number of States	WoLF-IGA	WoLF-PHC	GraWoLF	Unlimited Players	Limited Players	Self-Play	Beyond Self-Play
2-Player/2-Action Matrix Games	✓		✓	✓	2	1	✓			✓		✓	✓
Matching Pennies	✓		✓		2	1		✓		✓		✓	✓
Rock-Paper-Scissors	✓		✓		2	1		✓		✓	✓	✓	✓
Colonel Blotto	✓		✓		2	1		✓		✓	✓	✓	✓
3-Player Matching Pennies	✓			✓	3	1		✓		✓		✓	
Gridworld		✓		✓	2	73		✓		✓		✓	
Grid Soccer		✓	✓		2	760		✓		✓		✓	✓
Goofspiel		✓	✓		2	10^{11}			✓		✓	✓	
Keepout		✓	✓		2	∞			✓		✓	✓	

Table 8.1: Summary of the explored domains. Results for WoLF-IGA and WoLF-PHC with unlimited players can be found in Chapter 4. Results for WoLF-PHC with limited players can be found in Chapter 5. Results for GraWoLF can be found in Chapter 6.

8.2 Directions for Future Work

This dissertation opens up new interesting directions for further research in multiagent learning. The demonstration of effective learning in challenging tasks requiring approximation and other agent limitations expands the potential applications of learning techniques in multiagent environments. There are also additional fundamental questions that this work lays the foundations for considering and answering. These questions fall into four broad categories.

8.2.1 Further Theoretical Analysis of WoLF

The theoretical analysis of the effect of WoLF on learning presented in Chapter 4 is restricted to two-player, two-action matrix games. The result is strong evidence for the power of the WoLF variable learning rate, and when combined with the corroborating empirical results on a far wider class of domains it is very compelling. An extended analysis of WoLF beyond two-player, two-action matrix games, would be enlightening. The same technique that was used for our analysis is not likely to be useful toward this goal, though, as the number of qualitatively different dynamics increases exponentially with players and actions.

There has been recent work by Zinkevich (2003) demonstrating that Generalized Infinitesimal Gradient Ascent (GIGA), an extension of IGA to n-action games, minimizes regret, as was defined in Section 3.2.2. Whether an extension of WoLF-IGA might be able to make a similar guarantee would be interesting. Work has already begun to build on our contributions of variable learning rates and WoLF, investigating alternative rules for varying the learning rate (Banerjee & Peng, 2002). These rules, which are similar to WoLF, may ultimately prove easier to analyze.

Subsequent to our introduction of WoLF, Conitzer and Sandholm (Conitzer & Sandholm, 2003a) contributed an algorithm that is proven to be rational and convergent in self-play for all matrix games. Their result demonstrates that such an algorithm is possible, although the practicality of their algorithm is unclear. The algorithm does not obviously generalize to stochastic games, and convergence is enforced through requiring the agent to play the equilibrium when observing a non-stationary opponent. This prevents the learning of something like a restricted equilibrium, which can arise in the common case of limited agents. However, their result is still important as an existence proof for what is achievable by a learning algorithm.

Finally, from a purely game theoretic perspective, WoLF-IGA is essentially a dynamic that learns best-responses but also converges to a Nash equilibrium in self-play, e.g., (Kalai & Lehrer, 1993). This is a goal actively being investigated within the game theory community. An interesting recent negative result (Hart & Mas-Colell, 2002) shows that no dynamic that adjusts its strategy only based on its own reward matrix and the other player's current strategy could achieve this goal. IGA is an example of such a dynamic and therefore cannot guarantee convergence to an equilibrium in self-play, as we saw in Chapter 4. WoLF-IGA on the other hand does not fall under this result, since by adjusting the magnitude of the gradient through comparison with the equilibrium policy or the average policy, it injects information related to the other players' payoffs. Whether WoLF or another similar simple injection of knowledge could achieve this best-response convergence goal is of interest to the field of game theory, with implications for multiagent learning as well.

8.2.2 Asymmetric Learning

The focus of this dissertation has been on learning in self-play, where all agents are using nearly identical or similar learning algorithms. This focus has been intentional in order to avoid a common implicit assumption that other agents are more limited and inherently inferior. This assumption usually manifests itself in the form of a non-learning opponent, or an opponent whose behavior can be easily modeled. Despite our emphasis on self-play, in Chapter 4 we examined situations outside of self-play to demonstrate the generality of this work. This brief examination could be the seed of a very interesting direction of future study on asymmetric learning.

There are a number of different dimensions along which asymmetric learning could be studied. One consideration is that of exploiting a particular learning algorithm. Can one devise a learning algorithm to exploit another particular learning algorithm or even a whole class of simpler algorithms? Can one devise an algorithm that cannot be exploited, which is strongly related to regret minimizing algorithms? Can such an algorithm also have well-behaved and desirable performance in self-play? These questions relate to another dimension for exploring asymmetric learning, which is the idea of algorithms themselves being in equilibrium. By specifying some formulation on rewards over time, one could compute the "value" of two algorithms, and then examine what it means for algorithms to be in equilibrium. Essentially, this amounts to considering algorithms themselves as behavioral strategies, as described in Section 2.4.6, and examining whether two algorithms are in equilibrium. This has recently been investigated in a very theoretical fashion (Brafman & Tennenholtz, 2002a) relying mostly on folk theorems, but thus far, the practical implications are unclear. A final interesting dimension is human-agent interaction, where a learning agent is interacting with a learning human. Human adaptation is of a completely different form than most agent learning algorithms. Human policy changes are frequently to policies that are qualitatively different, as opposed to learning through slight variations. The amount of training required for human learning is also substantially smaller. These differences from agent-agent learning may have a major impact on the application of learning to such settings.

8.2.3 Evaluation

The goals and methods of evaluation of a multiagent learning algorithm are still very much open research questions. Evaluation becomes considerably more complicated when considering agent limitations. Since limitations may prevent equilibria from existing, convergence as a property and evaluation tool is useless. This raises the question of what should learning agents learn if equilibria do not exist? In Chapter 6, we presented and used three methods of evaluation that do not depend on equilibria or convergence. Other evaluation tools or methods need to be investigated and compared.

Another complication that multiagent domains add to the learning problem is that there are multiple components to an agent's performance. Effectiveness is determined by how adapted the agent's behavior is to the environment, as well as, how adapted the agent's behavior is to the opponent. In some games these two components are identical. For example, in Chess, we expect improving play against one opponent will improve play against all opponents.[1] Specifically, learning is focused on the rules of the game, or learning moves with guaranteed wins, or strong board positions independent of how the opponent plays. On the other hand, in rock-paper-scissors, this is not the case. In this game, learning to improve against a particular player's strategy necessarily causes one to play more poorly against another. Specifically, there is no need to adapt to the environment, performance depends only on adapting to the opponent. Most realistic and non-abstract problems are somewhere between these two extremes. The two learning tasks in Chapter 6, Goofspiel and Keepout, are examples of tasks that are somewhere in the middle of the continuum, between *all-domain* and *all-opponent.*

Understanding this domain-opponent continuum may be helpful for both designing and evaluating algorithms. Different domains may require different algorithms and different methods of evaluation. It also may be useful to evaluate how well an algorithm is adapted to the environment independently of how well the algorithm is adapted to the opponent. Challengers or improvement over the initial policies are evaluation methods more suited toward measuring how suited the agent's policy is to the environment, in general. On the other hand, a comparison of expected value of policies while learning may be more suited for measuring how well the agent is adapted to the specific opponent. More suited evaluation criteria may be able to distinguish these two components further. In addition, evaluation may depend on where the domain falls on the continuum of domain versus opponent. This continuum is not just important for adversarial domains, but should also give important insight into both collaborative and other non-competitive environments.

8.2.4 Training Efficiency

A final area of further research is that of training efficiency. How can an agent learn more quickly with less training experience? Although we believe the techniques presented in this work are

[1]This is not strictly true for Chess, but it is more true than in other settings due to the nature of alternating moves.

relatively competitive in terms of learning speed, we also recognize the need for using less data. Our learning in the Keepout task involved 4,000 trials, which is a small number by reinforcement learning standards. This is still far too slow for adapting to a particular opponent in the course of a soccer game, for example. The problem of learning with less data is shared with single-agent learning tasks, and single-agent solutions also can be applied to multiagent tasks. Parameterized policies, function approximation, eligibility traces, temporally abstract actions, and SMDPs are single-agent solutions that have all been incorporated into our applications of GraWoLF described in Chapter 6.

Multiagent tasks have an additional challenge or advantage related to training efficiency. How does one reuse training data between opponents? This problem is highly related to the continuum of games described in Section 8.2.3. Games like rock-paper-scissors offer little hope in the area of reusing training data, while in problems with no opponent, all of the training data can be used. The interesting problems are the cases between the extremes. For example, if the defender's action space in the Keepout task were changed so that it could navigate to a completely different set of points, how relevant would previous training be for the new policy? One simple solution would be to use the previously learned policy as the initial policy. More informed approaches of reuse is an interesting open problem.

8.3 Summary

This dissertation contributes a number of powerful techniques for learning in the presence of other learning agents. The cornerstone of the presented learning mechanisms is the WoLF principle for varying the learning rate and is widely applicable. We demonstrated effective learning in domains without state, in domains with small easily enumerable state spaces, and for complex tasks with intractably large or continuous state spaces, where agents have a variety of physical and rational limitations. We also demonstrated how learning could be incorporated in a coordinated team of agents to adapt to an unknown opponent. In addition, our analysis of the effect of limitations on the concept of equilibria is an important contribution with implications broader than just this dissertation's approach to multiagent learning. The thesis contributions provide both practical algorithms and foundational theory for learning to be applied in domains with ever increasing agent interaction. SDG[2]

[2]Johann Sebastian Bach finished many of his compositions with these initials. I echo his sentiments. See "http://www.bachfaq.org" for more information.

Appendix A

Learning Parameters

The following are the learning and decay rates used for the results presented throughout this dissertation.

Chapter 4

These are the learning rate parameters used with the algorithms: PHC, presented in Table 4.1, and WoLF-PHC, presented in Table 4.2. For all of the values, t refers to the number of times the particular state-action pair, (s, a), has been selected in Step 2(a). This requires maintaining a separate count for each pair. The learning rates used in matching pennies and rock-paper-scissors are purposefully very small so as to better illustrate graphically the dynamics of the learning algorithms.

Matching Pennies (Section 4.4.1)

$\alpha = \frac{1}{100+\frac{t}{10000}}$ $\delta = \delta_w = \frac{1}{20000+t}$ $\delta_l = 2\delta_w$

Rock-Paper-Scissors (Sections 4.4.1 and 4.4.5)

$\alpha = \frac{1}{10+\frac{t}{10000}}$ $\delta = \delta_w = \frac{1}{20000+t}$ $\delta_l = 2\delta_w$

Gridworld (Section 4.4.2)

$\alpha = \frac{1}{1+\frac{t}{500}}$ $\delta = \delta_w = \frac{1}{1000+\frac{t}{10}}$ $\delta_l = 4\delta_w$

Soccer (Sections 4.4.3 and 4.4.5)

$\alpha = \frac{1}{1+\frac{t}{500}}$ $\delta = \delta_w = \frac{1}{1+\frac{t}{10}}$ $\delta_l = 4\delta_w$

Three-Player Matching Pennies (Section 4.4.4)

$\alpha = \frac{1}{10+\frac{t}{10000}}$ $\delta = \delta_w = \frac{1}{100+t}$ $\delta_l = 2\delta_w$

Chapter 5

These are the learning rate parameters used with the algorithms, WoLF-PHC, presented in Table 4.2. As above, for all of the values, t refers to the number of times the particular state-action pair, (s, a), has been selected in Step 2(a). This requires maintaining a separate count for each pair. The learning rates used are purposefully very small so as to better illustrate graphically the dynamics of the learning algorithms.

Rock-Paper-Scissors (Section 5.3.1)

$\alpha = \frac{1}{10+\frac{t}{10000}}$ $\delta_w = \frac{1}{100+t}$ $\delta_l = 2\delta_w$

Colonel Blotto (Section 5.3.2)

$\alpha = \frac{1}{10+\frac{t}{10000}}$ $\delta_w = \frac{1}{20000+t}$ $\delta_l = 2\delta_w$

Chapter 6

These are the learning rate parameters used with the GraWoLF algorithm, presented in Table 6.1. For all of the values, t refers to the number of times Step 3 has been repeated.

Goofspiel (Section 6.4.1)

$\lambda = 0$ $\alpha = 0.2$ $\delta_l = 0.16$ $\delta_w = 0.008$ $\beta = \frac{1}{1+\frac{t}{4}}$

Keepout (Section 6.4.2)

$\lambda = 0.5$ $\alpha = \frac{0.1}{1+\frac{t}{200000}}$ $\delta_l = \frac{200}{1+\frac{t}{20000}}$ $\delta_w = \frac{10}{1+\frac{t}{20000}}$ $\beta = \frac{1}{1+\frac{t}{300}}$

Bibliography

Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences*, *10*, 25–61.

Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. E. (1995). Gambling in a rigged casino: The adversarial multi-arm bandit problem. In *36th Annual Symposium on Foundations of Computer Science*, pp. 322–331, Milwaukee, WI. IEEE Computer Society Press.

Baird, L. C., & Moore, A. W. (1999). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*. MIT Press.

Banerjee, B., & Peng, J. (2002). Convergent gradient ascent in general-sum games. In *13th European Conference on Machine Learning*. Springer Verlag.

Baxter, J., & Bartlett, P. L. (2000). Reinforcement learning in POMDP's via direct gradient ascent. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 41–48, Stanford University. Morgan Kaufman.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Blum, A., & Burch, C. (1997). On-line learning and the metrical task system problem. In *Tenth Annual Conference on Computational Learning Theory*, Nashville, TN.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on the Theoretical Aspects of Rationality and Knowledge*, pp. 195–210, Amsterdam, Netherlands.

Bowling, M. (2000). Convergence problems of general-sum multiagent reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 89–94, Stanford University. Morgan Kaufman.

Bowling, M., & Veloso, M. (1999). Bounding the suboptimality of reusing subproblems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 1340–1345, Stockholm, Sweden. Morgan Kaufman. An earlier version appeared in the *Proceedings of the NIPS Workshop on Abstraction in Reinforcement Learning*, 1998.

Bowling, M., & Veloso, M. (2001a). Convergence of gradient dynamics with a variable learning rate. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 27–34, Williams College.

Bowling, M., & Veloso, M. (2001b). Rational and convergent learning in stochastic games. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pp. 1021–1026, Seattle, WA.

Bowling, M., & Veloso, M. (2002a). Multiagent learning using a variable learning rate. *Artificial Intelligence*, *136*, 215–250.

Bowling, M., & Veloso, M. (2002b). Scalable learning in stochastic games. In *AAAI Workshop on Game Theoretic and Decision Theoretic Agents*, Edmonton, Canada.

Bowling, M., & Veloso, M. M. (2002c). Existence of multiagent equilibria with limited agents. Technical report CMU-CS-02-104, Computer Science Department, Carnegie Mellon University.

Brafman, R. I., & Tennenholtz, M. (2002a). Efficient learning equilibrium. In *Advances in Neural Information Processing Systems 14*. MIT Press.

Brafman, R. I., & Tennenholtz, M. (2002b). R-max, a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*, 213–231.

Brown, G. W. (1949). Some notes on computation of games solutions. Rand report P-78, The RAND Corporation, Santa Monica, California.

Browning, B., Bowling, M., & Veloso, M. (2002). Improbability filtering for rejecting false positives. In *Proceedings of the International Conference on Robotics and Automation*, Washington, D.C.

Browning, B., & Tryzelaar, E. (2003). Ubersim: A multi-robot simulator for robot soccer. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*.

Bruce, J., & Veloso, M. (2002). Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, pp. 2383–2388, Switzerland.

Carmel, D. (1997). *Model-based Learning of Interaction Strategies in Multi-agent Systems*. Ph.D. thesis, Computer Science Department, Technion.

Carmel, D., & Markovitch, S. (1996). Learning models of intelligent agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, CA. AAAI Press.

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Menlo Park, CA. AAAI Press.

Conitzer, V., & Sandholm, T. (2003a). AWESOME: a general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *Proceedings of the Twentieth International Conference on Machine Learning*. To appear.

Conitzer, V., & Sandholm, T. (2003b). Complexity results about Nash equilibria. In *Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico. To appear.

Filar, J., & Vrieze, K. (1997). *Competitive Markov Decision Processes*. Springer Verlag, New York.

Flood, M. (1985). Interview by Albert Tucker. The Princeton Mathematics Community in the 1930s, Transcript Number 11.

Fudenberg, D., & Levine, D. K. (1999). *The Theory of Learning in Games*. The MIT Press.

Gaughan, E. D. (1993). *Introduction to Analysis, 4th Edition*. Brooks/Cole Publishing Company, Pacific Grove, CA.

Gilboa, I., & Zemel, E. (1989). Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*.

Gilboa, I., & Samet, D. (1989). Bounded versus unbounded rationality: The tyranny of the weak. *Games and Economic Behavior*, 213–221.

Gintis, H. (2000). *Game Theory Evolving*. Princeton University Press.

Gordon, G. (2000). Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems 12*. MIT Press.

Greenwald, A., & Hall, K. (2002). Correlated Q-learning. In *Proceedings of the AAAI Spring Symposium Workshop on Collaborative Learning Agents*.

Hart, S., & Mas-Colell, A. (2002). Uncoupled dynamics cannot lead to nash equilibrium. Tech. rep. DP-299, The Hebrew University of Jerusalem, Center for Rationality.

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*.

Hoffman, A., & Karp, R. (1966). On nonterminating stochastic games. *Management Science*, *12*, 359–370.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press.

Hu, J. (1999). *Learning in Dynamic Noncooperative Multiagent Systems*. Ph.D. thesis, Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI.

Hu, J., & Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 242–250, San Francisco. Morgan Kaufman.

Jaakkola, T., Singh, S. P., & Jordan, M. I. (1994). Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 6*. MIT Press.

Jafari, A., Greenwald, A., Gondek, D., & Ercal, G. (2001). On no-regret learning, fictitious play, and nash equilibrium. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 226–223.

Jansen, P. J. (1992). *Using Knowledge About the Opponent in Game-tree Search*. Ph.D. thesis, Computer Science Department, Carnegie Mellon University.

Jehiel, P., & Samet, D. (2001). Learning to play games in extensive form by valuation. *NAJ Economics, Peer Reviews of Economics Publications*, *3*.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kalai, E., & Lehrer, E. (1993). Rational learning leads to nash equilibrium. *Econometrica*, *61*, 1019–1045.

Kitano, H., Kuniyoshi, Y., Noda, I., Asada, M., Matsubara, H., & Osawa, E. (1997). RoboCup: A challenge problem for AI. *AI Magazine*, *18*(1), 73–85.

Kuhn, H. W. (Ed.). (1997). *Classics in Game Theory*. Princeton University Press.

Larson, K., & Sandholm, T. (2001). Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, *132(2)*, 183–217.

Littlestone, N., & Warmuth, M. (1994). The weighted majority algorithm. *Information and Computation*, *108*, 212–261.

Littman, M. (2001). Friend-or-foe Q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 322–328. Morgan Kaufman.

Littman, M., & Stone, P. (2001). Leading best-response strategies in repeated games. In *Seventeenth Annual International Joint Conference on Artificial Intelligence Workshop on Economic Agents, Models, and Mechanisms*.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 157–163. Morgan Kaufman.

Littman, M. L., & Szepesvári, G. (1996). A generalized reinforcement-learning model: Convergence and applications. In *Proceedings of the 13th International Conference on Machine Learning*, pp. 310–318, Bari, Italy. Morgan Kaufmann.

Mangasarian, O. L., & Stone, H. (1964). Two-person nonzero-sum games and quadratic programming. *Journal of Mathematical Analysis and Applications*, *9*, 348–355.

Mannor, S., & Shimkin, N. (2001). Adaptive strategies and regret minimization in arbitrarily varying markov environments. In *Proceedings of the Fourteenth Annual Conference on Computational Learning Theory*, pp. 128–142.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, San Francisco. Morgan Kaufman.

McGovern, A., & Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 361–368. Morgan Kaufman.

McKelvey, R., & McLennan, A. (1996). Computation of equilibria in finite games. In *The Handbook of Computational Economics, Vol. I*, pp. 87–142. Elsevier.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, *13*, 103–130.

Nash, Jr., J. F. (1950). Equilibrium points in n-person games. *PNAS*, *36*, 48–49. Reprinted in (Kuhn, 1997).

Noda, I., Matsubara, H., Hiraki, K., & Frank, I. (1998). Soccer server: a tool for research on multi-agent systems. *Applied Artificial Intelligence*, *12*, 233–250.

Osborne, M. J., & Rubinstein, A. (1994). *A Course in Game Theory*. The MIT Press.

Peng, J., & Williams, R. J. (1996). Incremental multi-step Q-learning. *Machine Learning*, *22*, 283–290.

Pollatschek, M., & Avi-Itzhak, B. (1969). Algorithms for stochastic games with geometrical interpretation. *Management Science*, *15*, 399–415.

Reinhard, H. (1987). *Differential Equations: Foundations and Applications*. McGraw Hill Text.

Riley, P., & Veloso, M. (2000). *On Behavior Classification in Adversarial Environments*, pp. 371–380. Springer-Verlag.

Riley, P., & Veloso, M. (2002). Planning for distributed execution through use of probabilistic opponent models. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, Toulouse, France.

Robinson, J. (1951). An iterative method of solving a game. *Annals of Mathematics*, *54*, 296–301. Reprinted in (Kuhn, 1997).

Rosen, J. B. (1965). Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica*, *33*, 520–534.

Ross, S. M. (1971). Goofspiel — The game of pure strategy. *Journal of Applied Probability*, *8*, 621–625.

Rubinstein, A. (1998). *Modeling Bounded Rationality*. The MIT Press.

Rummery, G., & Niranjan, M. (1994). On-line q-learning using connectionisht systems. Tech. rep. Technical Report No. 166, University of Cambridge, Engineering Department.

Russell, S. (1997). Rationality and intelligence. *Artificial Intelligence*, *94*, 57–77.

Sandholm, T., & Crites, R. (1996). Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, *37*, 147–166.

Sen, S., Sekaran, M., & Hale, J. (1994). Learning to coordinate without sharing information. In *Proceedings of the 13th National Conference on Artificial Intelligence*.

Shapley, L. S. (1953). Stochastic games. *PNAS*, *39*, 1095–1100. Reprinted in (Kuhn, 1997).

Simon, H. A. (1982). *Models of Bounded Rationality*. MIT Press, Cambridge, MA.

Singh, S., Jaakkola, T., Littman, M. L., & Szepesvári, C. (2000a). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*.

Singh, S., Kearns, M., & Mansour, Y. (2000b). Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 541–548. Morgan Kaufman.

Sleator, D. D., & Tarjan, R. E. (1985). Self-adjusting binary search trees. *Journal of the ACM*, *32*, 652–686.

Stone, P., & Sutton, R. (2001). Scaling reinforcement learning toward Robocup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 537–534. Morgan Kaufman.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning*. MIT Press.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pp. 1057–1063. MIT Press.

Sutton, R. S., Precup, D., & Singh, S. (1998). Intra-option learning about temporally abstract actions. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 556–564, San Francisco. Morgan Kaufman.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 330–337, Amherst, MA.

Uther, W., & Veloso, M. (1997). Adversarial reinforcement learning. Tech. rep., Carnegie Mellon University. Unpublished.

Uther, W. T. B. (2002). *Tree Based Hierarchical Reinforcement Learning*. Ph.D. thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. Available as technical report CMU-CS-02-169.

Van der Wal, J. (1977). Discounted Markov games: successive approximations and stopping times. *International Journal of Game Theory*, *6*, 11–22.

von Neumann, J., & Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. John Wiley and Sons.

Vrieze, O. J. (1987). *Stochastic Games with Finite State and Action Spaces*. No. 33 in CWI Tracts. Centrum voor Wiskunde en Informatica.

Wang, X., & Sandholm, T. (2002). Reinforcement learning to play an optimal nash equilibrium in team markov games. In *Advances in Neural Information Processing Systems 14*. MIT Press.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, King's College, Cambridge, UK.

Weibull, J. W. (1995). *Evolutionary Game Theory*. The MIT Press.

Williams, R. J., & Baird, L. C. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Technical report, College of Computer Science, Northeastern University.

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning*. To appear.