Approaches to multi-agent learning

by

Yu-Han Chang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[June 2005]
May 2005

Author ....
Department of Electrical Engineering and Computer Science
May 20, 2005

Certified by .... [illegible] ......
Leslie Pack Kaelbling
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .... [illegible] ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

[redacted]

Approaches to multi-agent learning

by

Yu-Han Chang

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Systems involving multiple autonomous entities are becoming more and more prominent. Sensor networks, teams of robotic vehicles, and software agents are just a few examples. In order to design these systems, we need methods that allow our agents to autonomously learn and adapt to the changing environments they find themselves in. This thesis explores ideas from game theory, online prediction, and reinforcement learning, tying them together to work on problems in multi-agent learning.

We begin with the most basic framework for studying multi-agent learning: repeated matrix games. We quickly realize that there is no such thing as an opponent-independent, globally optimal learning algorithm. Some form of opponent assumptions must be necessary when designing multi-agent learning algorithms. We first show that we can exploit opponents that satisfy certain assumptions, and in a later chapter, we show how we can avoid being exploited ourselves.

From this beginning, we branch out to study more complex sequential decision making problems in multi-agent systems, or stochastic games. We study environments in which there are large numbers of agents, and where environmental state may only be partially observable. In fully cooperative situations, where all the agents receive a single global reward signal for training, we devise a filtering method that allows each individual agent to learn using a personal training signal recovered from this global reward. For non-cooperative situations, we introduce the concept of hedged learning, a combination of regret-minimizing algorithms with learning techniques, which allows a more flexible and robust approach for behaving in competitive situations. We show various performance bounds that can be guaranteed with our hedged learning algorithm, thus preventing our agent from being exploited by its adversary. Finally, we apply some of these methods to problems involving routing and node movement in a mobilized ad-hoc networking domain.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor of Computer Science and Engineering

Acknowledgments

I am indebted to my advisor Leslie Kaelbling for her vast stores of wisdom, her generous capability for finding time to share it, her invaluable guidance and encouragement, and her infinite patience while I searched for my research bearings. I would also like to thank my thesis committee members, Tommi Jaakola, Michael Littman, and John Tsitsiklis, for their helpful advice and insights. This thesis would not have been possible without the enduring support of my parents, who have encouraged me in the pursuit of knowledge and wisdom throughout my life, and Tracey Ho, who has been my love, light, and inspiration throughout the writing of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Learning enables us to adapt to the environment that surrounds us. By designing learning algorithms for robots and intelligent agents, we enable them to adapt to the environment into which we place them. This endeavor has produced many insights into the theory and practice of a single-agent learning to behave in a stationary environment. But what if the environment reacts to the agent? What if the environment includes opponents, or other agents? How do we need to change our learning methods to cope with a non-stationary or reactive environment?

Multi-agent learning captures an essential component of learning: the adaptivity of a learning agent to the environment around the agent. Often when we speak of learning in a single-agent domain, we implicitly assume that the environment is stationary. Our main goal in this single-agent case is to find a good policy for operating in this stationary environment. Due to the environment's stationarity, the speed with which we learn becomes secondary in importance to the eventual convergence of our learning algorithm to a near-optimal solution. Accordingly, while many existing reinforcement learning methods can be shown to find optimal policies in the limit, they are fairly slow to converge to this optimal policy.

In contrast, in multi-agent domains, the environment is assumed to change rapidly. Our opponents are part of the environment, and they may adapt to the way we act. They may indeed be trying to adapt to us as quickly as we are adapting to them. Thus, our learning algorithms must emphasize speed and adaptivity. This changes the focus in the design of good multi-agent learning algorithms from convergence in the long run to good performance in the short run. The algorithms must adapt quickly but still be able to achieve good performance within some given amount of time.

1.1 Motivation

Bees dance to announce the finding of food. Ants count the number of run-ins with other ants to determine their best current role, be it forager, maintenance, or drone. Antelope congregate for safety from hunting lions. Chimpanzees can learn by observing the mistakes of fellow chimps. And humans can acquire new behaviors by a variety of different means.

Whether these behaviors are learned or innate, they are invaluable for each species' survival in a world populated by a multitude of other animals. Either the species as a whole benefits from a cooperative behavior, or the individual benefits by avoiding a mistake committed by someone else or perhaps learning a new skill invented by someone else. While few people are prepared to call computer programs and robots a new species, various computer programs will eventually populate cyberspace, and increasingly sophisticated robots will begin to appear in the physical world around us. As a research community, we have thus far devoted most of our time creating intelligent agents or robots (henceforth called artificial agents, or simply agents) that can operate in various human-centric but static domains, whether directly helping humans in roles such as personal digital assistants or supporting human society in forms such as robotic assembly line manufacturers. However, as artificial agents become more prevalent and more capable, we may begin to see problems with this focus on the single agent. The interactions between all of the created agents will need to become a primary focus of our attention. The goals of the created agents may begin to conflict with one another, or we may begin to see the potential for cooperative behaviors amongst our agents.

At the most mundane level, consider a future fleet of autonomous vehicles, each delivering people or goods to prespecified locations. Most likely there will be some traffic information service that provides updates on traffic conditions for all of the possible routes for each vehicle. If all the vehicles acted selfishly, choosing the shortest route, the road system might experience bad oscillations in traffic volume. Ideally, we could enable the artificial agents in these scenarios to learn to cooperate with one another, settling on a policy of action that would be mutually beneficial. If we were a transit agency, our goal might be the overall efficiency of the entire regional road system. Or, if we were a luxury car manufacturer, our goal might be to design an agent with the single selfish aim of getting to its precious occupant to his or her destination as quickly as possible. Even in this case, the agent would have to rely on its prediction of the actions of all the other vehicles in the road network in order to achieve its desired goal. It could not operate as if the world were static, as if it were the only adaptive agent operating on the roads.

These are only two possible scenarios within the wide range of situations that might be encountered by a system of multiple agents. They are perhaps the most salient because it is easy to imagine that we will have to deal with these situations in the not-so-distant future. As such, these two scenarios represent the two main types of multi-agent systems we will explore further in this thesis. We call the second type "competitive, multi-designer systems" since the agents have conflicting goals and may be designed by many different individuals, each trying to create an agent that outsmarts all the others. In addition to purely competitive systems, we will also explore situations where coordination among the agents might lead to higher payoffs. The other type of situation is the "cooperative, single-designer system." In particular, we will explore complex cooperative systems that can be trained via simulation. They may or may not adapt further when actually deployed. The competitive agents, on the other hand, may or may not be trained prior to deployment, but will necessarily have to adapt their behavior in response to opponents once deployed and operating.

1.2 The approach

How do we begin studying such systems? Multi-agent domains are by their very nature highly complex. In order to study these domains, we need to pare them down to their bare essentials. One such setup is the repeated normal-form game. In this framework, the environment is the simplest it can be: a set of $N \times N$ matrices representing the rewards received by each agent, given a choice of actions by each agent. This allows us to focus on the game dynamics in terms of the interactions of the players, rather than their interaction with the environment. Most of the complexity will stem from the changing nature of the opponent, and the constant tension between trying to model and exploit the opponent while preventing the opponent from doing the same to us.

We will explore this framework in Chapter 3. Among other results, we will demonstrate that our beliefs about the types of opponents we are likely to face are very important for designing good algorithms. Some of this work was previously described by Chang and Kaelbling [2002].

1.3 Scaling up

From here, we try to increase the complexity of the environment, since it is hard to imagine repeated normal-form games finding wide applicability in real-world settings. Most real situations involve more complex environments that change as we act within them. Thus, our actions should affect the evolution of the environment state. In repeated normal-form games, we are always in the same state, playing the same normal-form game.

Furthermore, the environment is often too large for us to observe everything at once. If there are many agents operating in the same environment, we may not be able to observe what everyone is doing at the same time. Thus, we need methods that deal with this partial observability. Our approach is somewhat different from the traditional approach in the literature on partially observable Markov decision processes (POMDPs), where methods have been developed to use memory to keep track of unobserved state variables. We also use a memory variable, but our main idea is to filter the reward signal so that a single agent is able to receive a training signal that is proportional to the credit that it is due. Thus, we do not explicitly track the states of the other agents, other than to maintain a single memory variable that summarizes the contribution of the rest of the system to the observed reward signal. Some of this work was previously described by Chang, Ho, and Kaelbling [2004a].

1.4 Highly complex domains

While the filtering approach provides us with a simple method for resolving multi-agent credit assignment in certain situations, its effectiveness is also limited. When the number of agents grows large, or if the agents are self-interested, the filtering approach cannot retrieve enough information from the observed reward signal to filter out a good training signal. Furthermore, the filtering method implicitly assumes that the environment, including the other agents, operates independently from the agent in question. In an adversarial domain, however, this is no longer the case, and thus the filtering approach would fail.

In such cases, we may need to develop a model of the world that explicitly includes the other agents. The states and potential actions of all the agents in the system would have to be considered in order to estimate the value of an observed state from one agent's perspective. However, as Bernstein et al. [2002] have shown, even if we allow the agents to explicit communicate their partial state observations periodically, the decentralized control of a Markov decision process (MDP) is intractable. More specifically, in this case, it can be proved that there cannot be any polynomial-time algorithms to accomplish this task. Given this predicament, we need to resort to approximations or further assumptions in order to design algorithms (and thus agents) that perform reasonably well in such domains. Policy search is one such approximation [Peshkin *et al.*, 2000].

Even worse, we can show that if we do not make any assumptions about the opponent at all, then it is impossible to construct a learning algorithm that is optimal over all possible opponents. We must either restrict our attention to certain classes of opponents, or we must define a new notion of optimality. In this thesis, we introduce a new algorithm that approximates traditional optimal performance against large classes of opponents, and guarantees a regret bound against arbitrary opponents, which can be considered as a new notion of optimality.

We consider the potential for using experts that may be able to provide us with advice about which actions to perform. We might hope to resort to the advice of these experts to help us focus our search on potentially fruitful policies. The problem we then face is deciding between different expert advice. This framework can be used to derive a variety of different learning algorithms that provide interesting guarantees. In these combined learning algorithms, we can consider an individual learning algorithm to be one of the experts that provides us with advice from what it has learned thus far. We will call this approach hedged learning, since the algorithm hedges between following the various experts. For example, we present an algorithm that combines a set of given experts with an efficient MDP learner to create an algorithm that provides both a regret-minimizing guarantee and a polynomially time-bounded guarantee that it will find a near-optimal policy for the MDP.

Using experts provides us with a way of dealing with the adversarial, changing nature of the opponents in competitive settings. This framework is able to give us worst-case guarantees against such opponents. As long as we have decent experts to listen to, our performance is guaranteed to at least mimic the best expert closely and thus should be reasonably good. The importance of using these regret-minimizing techniques is that it insures that our learning process is in some sense *hedged* . That is, if one of our learning algorithms fails, we have intelligently hedged our bets between multiple different experts and thus we limit our losses in these cases. Some of this work is described by Chang and Kaelbling [2005].

1.5 A specialization: cooperative, single-designer systems

We next study what seems to be the easier scenario: all of the agents are designed by a single architect, and all of the agents are working towards a single global goal. There is no longer a competitive issue caused by adversarial opponents. By limiting ourselves to systems where a single designer controls the entire system, we also do not need to consider interoperability issues and goal negotiation or communication amongst the agents. All of the agents know what the goal is and cooperate to achieve the goal.

Moreover, we might imagine that the designer is able to construct an approximate simulation of the world in which the agents are being designed to operate. These simulators could thus be used to train the agents before they are actually deployed in real situations. Flight simulators accomplish much the same task for human pilots.

We will introduce the mobilized ad-hoc networking domain as a basis for motivating our ideas on creating effective learning algorithms for cooperative multi-agent systems. This domain will also serve as a test-bed for our algorithms. Some of this work was previously described in Chang, Ho, and Kaelbling [2003].

1.5.1 Mobilized ad-hoc networking

Mobile ad-hoc networking is emerging as an important research field with a number of increasingly relevant real-world applications, ranging from sensor networks to peer-to-peer wireless computing. Researchers in AI and machine learning have not yet made major contributions this growing field. It promises to be a rich and interesting domain for studying the application of learning techniques. It also has direct applications back to AI, for example in the design of communication channels for groups of robots. We introduce mobilized ad-hoc networks as a multi-agent learning domain and discuss some motivations for this study. Using standard reinforcement learning techniques, we tackled two distinct problems within this domain: packet routing and node movement. We demonstrate that relatively straightforward adaptations of these methods are capable of achieving reasonable empirical results. Using the more sophisticated techniques we develop in Chapter 5, we are able to improve upon this performance.

Mobile ad-hoc networks have not traditionally been considered a multi-agent learning domain partly because most research in this area has assumed that we have no control over the node movements, limiting research to the design of routing algorithms. Each node is assumed to be attached to some user or object that is moving with its own purpose, and routing algorithms are thus designed to work well under a variety of different assumptions about node mobility patterns.

However, there are many applications in which we can imagine that some of the nodes would have control over their own movements. For example, mobile robots might be deployed in search-and-rescue or military reconnaissance operations requiring ad-hoc communication. In such cases it may be necessary for some nodes to adjust their physical positions in order to maintain network connectivity. In these situations, what we will term *mobilized* ad-hoc networks becomes an extremely relevant multi-agent learning domain. It is interesting both in the variety of learning issues involved and in its practical relevance to real-world systems and technology.

There are several advantages gained by allowing nodes to control their own movement. Stationary or randomly moving nodes may not form an optimally connected network or may not be connected at all. By allowing nodes to control their own movements, we will show that we can achieve better performance for the ad-hoc network. One might view these controllable nodes as "support nodes" whose role is to maintain certain network connectivity properties. As the number of support nodes increases, the network performance also increases. Given better movement and routing algorithms, we can achieve significant additional performance gains.

It is important to note that there are two levels at which learning can be applied: (1) packet routing and (2) node movement. We will discuss these topics in separate sections, devoting most of our attention to the more difficult problem of node movement. Packet routing concerns the forwarding decisions each node must make when it receives packets destined for some other node. Node movement concerns the actual movement decisions each node can make in order to optimize the connectivity of the ad-hoc network. Even though we will use reinforcement learning techniques to tackle both these problems, they must each be approached with a different mind set. For the routing problem, we focus on the issue of online adaptivity. Learning is advantageous because it allows the nodes to quickly react to changes in network configuration and conditions. Adaptive distributed routing algorithms are particularly important in ad-hoc networks, since there is no centrally administered addressing and routing system. Moreover, network configuration and conditions are by definition expected to change frequently.

On the other hand, the node movement problem is best handled off-line. Learning a good movement policy requires a long training phase, which would be undesirable if done on-line. At execution time, we should simply be running our pre-learned optimal policy. Moreover, this movement policy should encode optimal action selections given different observations about the network state; the overall policy does not change due to changing network configuration or conditions and thus does not need to adapt online. We treat the problem as a large partially-observable Markov decision process (POMDP) where the agent nodes only have access to local observations about the network state. This partial observability is inherent to both the routing and movement portions of the ad-hoc networking problem, since there is no central network administrator. Nodes can only observe the local state around them; they do not have access to the global network topology or communication patterns. Even with this limited knowledge, learning is useful because it would otherwise be difficult for a human designer to create an optimized movement policy for each network scenario.

Within this POMDP framework, we first attempt to apply off-the-shelf reinforcement learning methods such as Q-learning and policy search to solve the movement problem. Even though Q-learning is ill-suited to partially-observable domains, and policy search only allows us to find a locally optimal policy, we show that the resulting performance is still reasonable given a careful construction of the observation space. We then apply more sophisticated techniques that combine multiple learning methods and predefined experts, resulting in much improved performance.

The packet routing and node movement problems represent two different methods for handling non-stationarity. In the case of packet routing, we allow the nodes to adapt rapidly to the changing environment. This type of fast, constant learning allows the nodes to maintain routing policies that work well, even though the environment is continuously changing. In the case of node movement, we take a different tack. We attempt to model the relevant aspects of the observed state space in order to learn a stationary policy. This stationary policy works well even though aspects of the environment constantly change, because we have modeled these changes in the policy's state space.

1.6 Thesis contributions

To summarize, this thesis presents new work in a diverse range of multi-agent settings. In the simplest repeated game setup, we present an analysis of the shortcomings of previous algorithms, and suggest improvements that can be made to the design of these algorithms. We present a possible classification for multi-agent algorithms, and we suggest new algorithms that fit into this classification.

In more complicated settings, we present a new technique that uses filtering methods to extract a good training signal for a learning agent trying to learn a good reactive policy in a cooperative domain where a global reward signal signal is available. If the agent is given expert advice, we show that we can combine MDP learning algorithms with online learning methods to create an algorithm that provides online performance guarantees together with polynomially time-bounded guarantees for finding a near-optimal policy in the MDP.

If the setting is not cooperative, we cannot apply this filtering technique. We introduce the idea of hedged learning, where we incorporate knowledge about potential opponents in order to try to play well against as many possible opponents as possible. We show performance guarantees that can be obtained using this approach.

Finally, we apply some of these methods to the problem of node movement and packet routing in mobilized ad-hoc networks.

Chapter 2

Background

It comes as no surprise that there have been a wide range of proposed algorithms and models to tackle the problems we discussed in the previous chapter. One such framework, the repeated game, was first studied by game theorists and economists, and later it was studied by the machine learning community interested in multi-agent games. For more complex games, our use of filtering methods follows a long line of applications of filtering techniques for tracking underlying states of a partially observed variable. Regret-minimizing methods have been extensively studied by the online learning community, but have also been studied under different names such as *universal consistency* by the game theory community and *universal prediction* (and universal codes) by the information theory community. In this chapter, we provide background information on these diverse but very related fields.

2.1 Repeated games

Repeated games form the simplest possible framework for studying multi-agent learning algorithms and their resulting behavior. We first lay out the mathematical framework and provide an introduction to the basic definitions of game theory, then give a classification of prior work in the field of multi-agent learning algorithms. We then discuss some of the prior work developed within this framework, from both the game theory and machine learning perspectives.

2.1.1 Mathematical setup

From the game theory perspective, the repeated game is a generalization of the traditional one-shot game, or *matrix game.* In the one-shot game, two players meet, choose actions, receive their rewards based on the simultaneous actions taken, and the game ends. Their actions and payoffs are often visually represented as a matrix, where action choices correspond to the rows or columns depending on the player executing the action, and payoffs are shown in the matrix cells. Specifically, the reward matrix R_i for each player i is defined as a function $R_i : A_1 \times A_2 \rightarrow \mathbb{R}$, where A_i is the set of actions available to player i. As discussed, R_1 is often written as an $|A_1| \times |A_2|$ matrix, with $R_1(i, j)$ denoting the reward for agent 1 if agent 1 plays action $i \in A_1$ and agent 2 plays action $j \in A_2$. The game is described as a tuple (A_1, A_2, R_1, R_2) and is easily generalized to n players.

Some common (well-studied) examples of two-player, one-shot matrix games are shown in Figure 2-1. The special case of a purely competitive two-player game is called a *zero-sum game* and must satisfy $R_1 = -R_2$.

In general, each player simultaneously chooses to play a particular pure action $a_i \in A_i$, or a mixed policy $\mu_i \in PD(A_i)$, where $PD(A_i)$ is the set of all possible probability distributions over the set of actions. The players then receive reward based on the joint actions taken. In the case of a pure action, the reward for agent i is given by $R_i(a_i, a_j)$, and in the case of mixed policies, the reward is given as an expectation, $R_i(\mu_i, \mu_j) = E[R_i(a_i, a_j)|a_i \sim \mu_i, a_j \sim \mu_j]$. For games where there are more than two players, we will sometimes write the joint actions of all the other agents as a_{-i} or μ_{-i}.

The literature usually uses the terms *policy* and *strategy* interchangeably. As we will see later, sometimes policies or strategies will be more complicated mappings between states and action distributions.

The traditional assumption is that each player has no prior knowledge about the other player. Thus there is no opportunity to tailor our choice of action to best respond to the opponent's predicted action. We cannot make any predictions. As is standard in the game theory literature, it is thus reasonable to assume that the opponent is fully rational and chooses actions that are in its best interest. In return, we would like to play a *best response* to the opponent's choice of action.

Definition 1 *A best response function for player i, $BR_i(\mu_{-i})$, is defined to be the set of all optimal policies for player i, given that the other players are playing the joint policy μ_{-i}: $BR_i(\mu_{-i}) = \{\mu_i^* \in M_i | R_i(\mu_i^*, \mu_{-i}) \geq R_i(\mu_i, \mu_{-i}), \forall \mu_i \in M_i\}$, where M_i is the set of all possible policies for agent i.*

If all players are playing best responses to the other players' strategies, then the game is said to be in *Nash equilibrium*.

Definition 2 *A Nash equilibrium is a joint policy μ such that for every agent i, $\mu_i \in BR_i(\mu_{-i})$.*

The fixed point result by Nash shows that every game must have at least one Nash equilibrium in mixed policies. Once all players are playing a Nash equilibrium, no single player has an incentive to unilaterally deviate from his equilibrium policy. Any game can be solved for its Nash equilibria using quadratic programming, and if the game only has a single Nash equilibrium, then a player can choose a strategy for playing against a similar, fully rational player in this fashion, given prior knowledge of the game structure. However, computing a Nash equilibrium is a computationally hard problem. To date there are no polynomial time algorithms for exactly solving a game to find all of its Nash equilibria, and it is an open question whether this problem is NP-hard [Papadimitriou, 2001].

A different problem arises when there are multiple Nash equilibria. If the players do not manage to coordinate on one equilibrium joint policy, then they may all end up worse off. The Hawk-Dove game shown in Figure 1(c) is a good example of this problem. The two Nash equilibria occur at (1,2) and (2,1), but if the players do not coordinate, they may end up playing a joint action (1,1) and receive 0 reward.

In the zero-sum setting, a fully rational opponent will choose actions that minimize our reward, since this in turn maximizes its reward. This leads to the concept

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

$$R_2 = -R_1$$

(a) Matching pennies

$$R_1 = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

$$R_2 = -R_1$$

(b) Rock-Paper-Scissors

$$R_1 = \begin{bmatrix} 0 & 3 \\ 1 & 2 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 0 & 1 \\ 3 & 2 \end{bmatrix}$$

(c) Hawk-Dove, a.k.a. "Chicken"

$$R_1 = \begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix}$$

(d) Prisoner's Dilemna

Figure 2-1: Some common examples of single-shot matrix games.

of minimax optimality, where we choose to maximize our reward given that the opponent is trying to minimize our reward. Von Neumann's minmax theorem proves the existence of an equilibrium point where both players are playing their minmax optimal strategies:

$$\max_i \min_j R_1(\mu_i, \mu_j) = \min_j \max_i R_1(\mu_i, \mu_j) = -\min_j \max_i R_2(\mu_i, \mu_j) \quad .$$

Traditionally, zero-sum games are also called "pure competitive" games. We use the term "competitive games" to include both pure, zero-sum games and general sum games where the agents have competing goals.

Minimax optimality in zero-sum games is a special case of *Nash equilibrium* in general-sum games. In general-sum games, we can no longer assume that the other player is trying to minimize our payoffs, since there are no assumptions about the relationship between the reward functions R_i. We must generally play a *best response* (BR) to the opponent's choice of action, which is a much harder question since we may be unable to predict the opponent's choice of action. We may need to know the opponent's utility function R_{-i}, and even if we know R_{-i}, the opponent may or may not be acting fully rationally. Moreover, to complicate matters further, the set of opponent policies that can be considered "fully rational" will ultimately depend on the space of opponent types we are willing to consider, as we will discuss later.

2.1.2 Prior work

Nash equilibrium is commonly used as a solution concept for one-shot games. In contrast, for repeated games there are a range of different perspectives. Repeated games generalize one-shot games by assuming that the players repeat the matrix game over many time periods. Researchers in reinforcement learning view repeated games as a special case of stochastic, or Markov, games. Researchers in game theory, on the other hand, view repeated games as an extension of their theory of one-shot matrix games. The resulting frameworks are similar, but with a key difference in their treatment of game history. Reinforcement learning researchers have most often focused their attention on choosing a single myopic stationary policy μ that will maximize the learner's expected rewards in all future time periods given that we are in time t, $\max_{\mu} E_{\mu}\left[\sum_{\tau=t}^{T} \gamma^{\tau-t} R^{\tau}(\mu)\right]$, where T may be finite or infinite, and $\mu \in PD(A)$. In the infinite time-horizon case, we often include the discount factor $0 < \gamma < 1$. We call such a choice of policy μ *myopic* because it does not consider the effects of current actions on the future behavior of the opponent.

Littman [1994] analyzes this framework for zero-sum games, proving convergence to the Nash equilibrium for his minimax-Q algorithm playing against another minimax-Q agent. Claus and Boutilier [1998] examine cooperative games where $R_1 = R_2$, and Hu and Wellman [1998] focus on general-sum games. These algorithms share the common goal of finding and playing a Nash equilibrium. Littman [2001] and Hall and Greenwald [2001] further extend this approach to consider variants of Nash equilibrium, called *correlated equilibria*, for which convergence can be guaranteed.

When the opponent is not rational, it is no longer advantageous to find and play a Nash equilibrium strategy. In fact, given an arbitrary opponent, the Nash equilibrium strategy may return a lower payoff than some other action. Indeed, the payoff may be even worse than the original Nash equilibrium value. For example, in the game of Hawk-Dove in Figure 2-1, one Nash equilibrium occurs when player 1 plays the first row action, and player 2 plays the second column action, leading to a reward of 3 for player 1 and a reward of 1 for player 2. However, if player 2 deviates and plays action 1 instead, they will both receive a reward of zero.

Even in zero-sum games, the problems still exists, though in a more benign form. As long as an agent plays a Nash equilibrium strategy, it can guarantee that it will receive payoff no worse than its expected Nash payoff, even if its opponent does not play its half of the Nash equilibrium strategy. However, even in this case, playing the Nash equilibrium strategy can still be suboptimal. For example, the opponent may not be playing a rational strategy. We would no longer need to worry about the mutual optimality condition of a Nash equilibrium; instead, we can simply play a best response to the opponent and exploit its irrationalities. Thus, in both general-sum and zero-sum games, we might hope that our agents are not only able to find Nash equilibrium strategies, but that they are also able to play best response strategies against non-Nash opponents.

Bowling and Veloso [2002b] and Nagayuki et al. [2000] propose to relax the mutual optimality requirement of Nash equilibrium by considering *rational* agents, which always learn to play a stationary best-response to their opponent's strategy, even if the opponent is not playing an equilibrium strategy. The motivation is that it allows our agents to act rationally even if the opponent is not acting rationally because of physical or computational limitations. Fictitious play [Fudenburg and Levine, 1995] is a similar algorithm from game theory.

Again, the goal is similar to the Nash approach: the agents should exhibit *convergence* to a stationary equilibrium, even if it is not the Nash equilibrium. Moreover, the agents should converge to a Nash equilibrium against a stationary Nash opponent, but should be able to exploit any weaknesses of a non-Nash player. Indeed, Bowling and Veloso show that in games where there is a single Nash equlibrium, a special full-knowledge version of their WoLF-PHC algorithm converges to the Nash equilibrium when playing against itself.

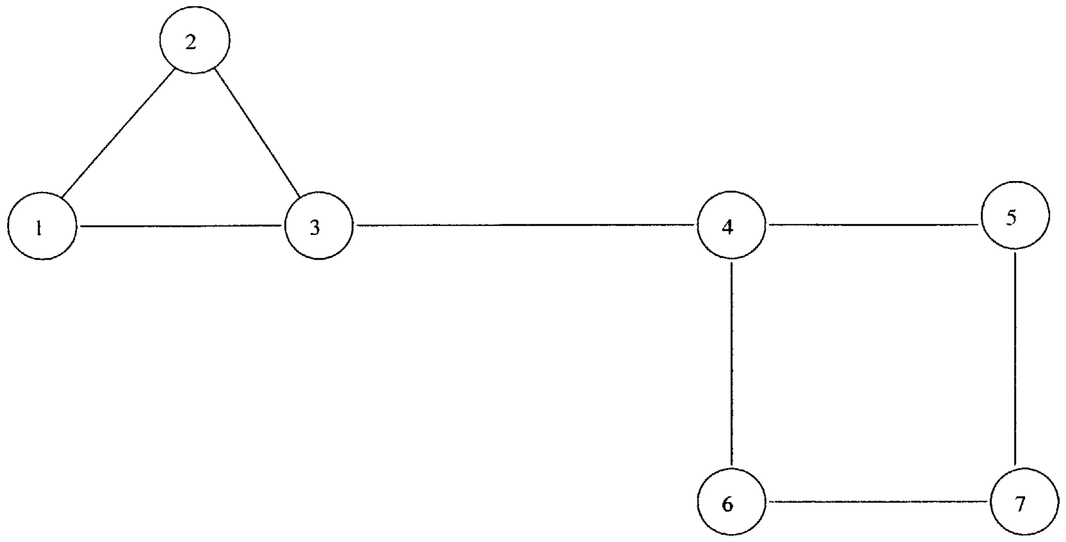


Figure 2-2: This figure shows an example of a network game. There are seven players in this game, and the links between the players represent relationships between players that results in direct payoff consequences. For example, Player 1's payoffs are only affected by its own actions, and the actions of Player 2 and Player 3. They are only indirectly affected by the action choices of the remaining four players, if Players 2 or Player 3's action choice is influenced by the rest of the players.

2.1.3 Solving for Nash equilibria

Recently there has been a great deal of interest in computational techniques for finding one or more Nash equilibria strategies for a given game. The general problem itself is usually quite difficult computationally, but it is an open question whether it is in P in the size of the game [Papadimitriou, 2001]. It is conjectured that the problem is somewhere between P and NP, since it is known to have at least one solution, unlike most problems that are NP-hard. However, new techniques take advantage of special structures in certain games to derive efficient computational techniques for finding Nash equilibria. For example, Littman et al. [2001] study network games, where a large game is structured into local interactions involving only small groups of agents. These local interactions may have effects that propagate throughout the network, but the immediate rewards of the local games are only directly observed by the local game participants. Figure 2-2 shows an example of a simple network game.

Using this structure, the authors are able to show that one can compute ϵ-

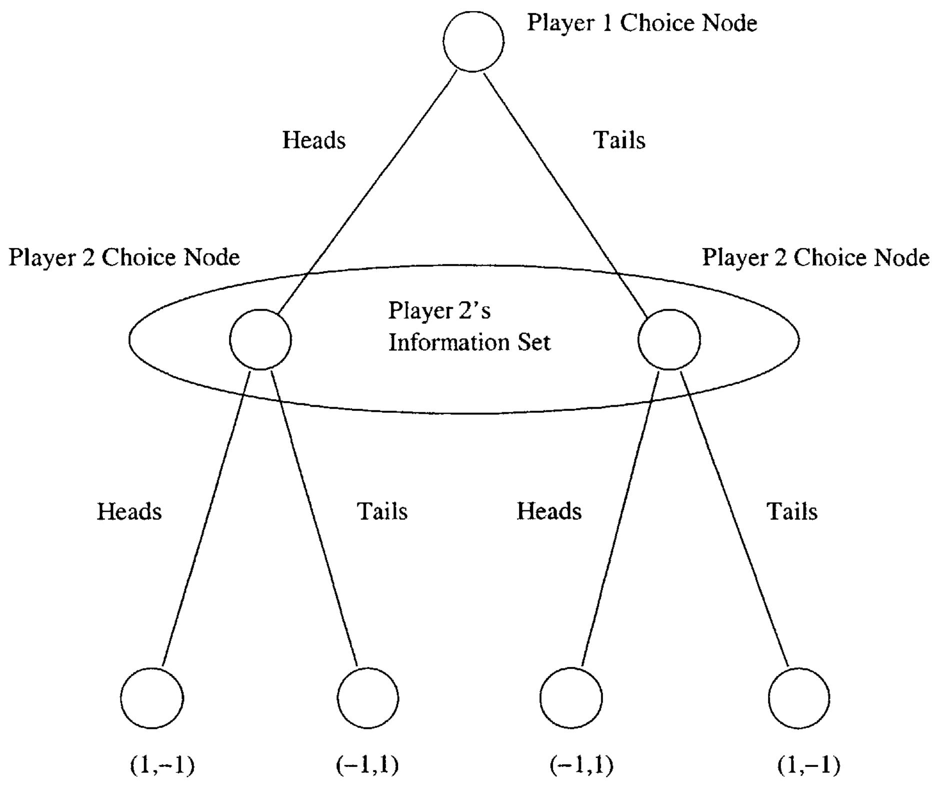


Figure 2-3: This figure shows the Matching Pennies game in extensive form. Clearly, the extensive form is a more general representation, since we can now allow the players to move sequentially and possess different information sets. Though not needed in this particular game, the extensive form representation can also include probabilistic choice nodes for "Nature", which can be used to account for randomness in the reward structure.

Nash equilibria in time polynomial in the size of the largest sub-game. Ortiz and Kearns [2002] extend this work to compute Nash equilibrium in loopy graphical games.

Blum, Shelton, and Koller study a more general framework, described by *multi-agent influence diagrams*, or MAIDs [Blum *et al.*, 2003]. The diagrams also graphically represent game structure, but they are richer descriptions than the network games studied by Kearns et al. MAIDs are able to capture information contained in extensive form games in a compact manner. Extensive form games generalize normal form, or matrix, games by allowing the representation of sequential moves by the various players and different information sets available to the various players. Extensive form games are usually represented as a tree, where branches of the tree represent the different actions and agent may take from a particular node of the tree (Figure 2-3). Blum et al. provide an algorithm that can compute an exact Nash equilibrium strategy in polynomial time.

2.1.4 Correlated Nash equilibria

For the sake of completeness, we will also mention the concept of a correlated Nash equilibrium. The correlated Nash equilibrium generalizes Nash equilibrium by allowing the players to coordinate, or correlate, their actions by observing a randomized external source of information. For example, consider this situation: two cars are approaching an intersection. If we assume that they must take simultaneous actions, then their Nash equilibrium would usually involve a randomized policy that chooses between the two possible actions, stop or go. However, if we add a traffic light to the intersection, then the cars can coordinate their actions based on the signal given by the traffic light. This would be a correlated Nash equilibrium.

It can be shown that if we force all the agents to use a particular learning algorithm, then the system will converge to a correlated Nash equilibrium [Greenwald and Hall, 2003]. The intuition is that this learning algorithm uses the observed history of players' actions as the coordination signal.

2.1.5 Game theoretic perspective of repeated games

As alluded to earlier, modern game theory often take a more general view of optimality in repeated games. The machine learning community has also recently begun adopting this view [Chang and Kaelbling, 2002; de Farias, 2004]. The key difference is the treatment of the history of actions taken in the game. Recall that in the stochastic game model, we took policies to be probability distributions over actions, $\mu_i \in PD(A_i)$. We referred to this choice of policy as a myopic choice, since it did not consider the effects of the current policy on the future behavior of the opponent.

Here we redefine a policy to be a mapping from all possible histories of observations to distributions over actions, $\mu_i : H \rightarrow PD(A_i)$, where $H = \bigcup_t H^t$ and H^t is the set of all possible histories of length t. Histories are observations of joint actions, $h^t = (a_i, a_{-i}, h^{t-1})$. Player i's strategy at time t is then expressed as $\mu_i(h^{t-1})$.

We can now generalize the concept of Nash equilibrium to handle this new space of possible strategies.

Definition 3 *Let h_t be a particular sequence of actions that can be observed given that the agents are following policies μ_i and μ_{-i}. Given the opponent's policy μ_{-i}, a best response policy for agent i maximizes agent i's reward over the duration t of the game. We define the set of best response policies to be $BR_i(\mu_{-i}) = \{\mu_i^* \in M_i | \sum_{\tau=1}^{t} R_i^\tau(\mu_i^*, \mu_{-i}) \geq \sum_{\tau=1}^{t} R_i^\tau(\mu_i, \mu_{-i}), \forall \mu_i \in M_i\}$, where M_i is the set of all possible policies for agent i and $R_i^\tau(\mu_i, \mu_{-i})$ is the expected reward of agent i at time period τ.*

A Nash equilibrium in a repeated game that is repeated for t time periods is a joint strategy $\mu(H)$ such that for every agent i, and every history of observations h^t that can be observed in the game given $\mu(H)$, $\mu_i(h^t) \in BR_i(\mu_{-i}(h^t))$.

For the infinitely repeated game, we let $t \to \infty$. Now, the set of best response policies $BR_i(\mu_{-i})$ becomes the set of all policies μ_i^* such that

$$\lim_{t\to\infty} \sup \sum_{\tau=1}^{t} \delta^{\tau-1} R_i^\tau(\mu_i^*, \mu_{-i}) \geq \lim_{t\to\infty} \sup \sum_{\tau=1}^{t} \delta^{\tau-1} R_i^\tau(\mu_i, \mu_{-i}), \quad \forall \mu_i \in M_i,$$

where $\delta \leq 1$ is a discount factor.

Let h be an infinite sequence of actions that can be observed given that the agents are following policies μ_i and μ_{-i}. A Nash equilibrium in the infinitely repeated game is a joint strategy $\mu(H)$ such that for every agent i, and every history of observations h^t that can be observed in the game given $\mu(H)$, $\mu_i(h) \in BR_i(\mu_{-i}(h))$.

This definition requires that *strategies* be best responses to one another, where the earlier definition only required that simple *probability distributions over action* be best responses to one another. Henceforth, we will refer to the equilibrium in the earlier definition as a Nash equilibrium of the one-shot game. It is a stationary policy that is myopic.

By allowing our agent to use policies that are conditioned on the observed history of play, we are endowing our agent with memory. Moreover, the agent ought to be able to form beliefs about the opponent's strategy, and these beliefs ought to converge to the opponent's actual strategy given sufficient learning time. Let $\beta_i : H \rightarrow PD(A_{-i})$ be player i's belief about the opponent's strategy. That is, given a particular history H, a belief β would map that history to a probability distribution over actions that we expect the opponent to play. In this sense, the belief mapping is similar to a policy mapping that maps states to action distributions. Then a learning path is defined to be a sequence of histories, beliefs, and personal strategies. Now we can define a Nash equilibrium of a repeated game in terms of our personal strategy and our beliefs about the opponent. If our prediction about the opponent's strategy is accurate, then we can choose an appropriate best-response strategy. If this holds for all players in the game, then we are guaranteed to be in Nash equilibrium [Greenwald and Hall, 2003].

Proposition 4 *A learning path* $\{(h^t, \mu_i(h^{t-1}), \beta_i(h^{t-1}))|t = 1, 2, \ldots\}$ *converges to a Nash equilibrium iff the following two conditions hold:*

- *Optimization:* $\forall t, \mu_i(h^{t-1}) \in BR_i(\beta_i(h^{t-1}))$. *We always play a best-response to our prediction of the opponent's strategy.*
- *Prediction:* $\lim_{t \rightarrow \infty} |\beta_i(h^{t-1}) - \mu_{-i}(h^{t-1})| = 0$. *Over time, our belief about the opponent's strategy converges to the opponent's actual strategy.*

However, Nachbar and Zame [1996] show that this requirement of simultaneous prediction and optimization is impossible to achieve, given certain assumptions about our possible strategies and possible beliefs. We can never design an agent that will learn to both predict the opponent's future strategy and optimize over those beliefs at the same time. Despite this fact, we can design an algorithm that approximates the best-response stationary policy over time against *any* opponent. In the game theory literature, this concept is often called *universal consistency.* In the online learning literature, it is often referred to as *regret minimization.*

2.2 Regret-minimization and online learning

This regret-minimization framework relaxes the setup of the repeated game framework. It no longer assumes that the environment (or opponent) is modeled by a matrix game at each time period. Instead, it simply assumes that the agent interacts with a black-box environment at each time period by choosing an action to play. The agent then receives a reward, and may or may not be able to observe the expected reward he would have received had he played any of his other possible actions. Thus, the black-box environment could be an opponent, a fixed finite automaton, or any general environment. For example, these methods have been applied to cases where the environment is a set of casino slot-machines (one-armed bandits) [Auer *et al.*, 1995], the stock market [Blum and Kalai, 1997], or even a heuristic measurement of robot arm positioning performance [Hsu *et al.*, 2005].

2.2.1 Prior work

The first work on regret-minimization came from two independent fields, again, game theory and machine learning. Fudenburg and Levine [1995] and Freund and Schapire [1995; 1999] independently show that two analogous techniques, exponential fictitious play and a multiplicative-weight algorithm, exhibit universal consistency or regret-minimization. One limitation of these algorithms is that they require the player to observe the expected rewards they would have received if they had played any of the available actions, rather than only the action they actually played. This usually requires knowledge of the opponent's mixed strategy, since we need to know the expected rewards. Clearly this is usually unavailable to us. Auer et al. [1995; 2002] extend Freund and Schapire's technique to the case where outcomes are only observed for the actions that are chosen, thereby getting around this problem while introducing a small additional penalty in the achievable performance bounds.

There have been numerous related articles, applying the basic technique of maintaining a mixture of weights to other areas. Cesa-Bianchi and Lugosi [2003] provide a general proof of this kind of potential-based algorithm and relate it to different variants of the technique that have been proposed. Their proof builds on the ideas of Hart and Mas-Colell's analysis [2001] of Λ-strategies for playing iterated games. Other authors have also extended the results to cases where the action space is continuous, or where different loss functions are assumed [Vovk, 1998; Mannor and Shimkin, 2003].

Interestingly, the machine learning technique of boosting is closely tied to this work. Freud and Schapire [1995] provide the basic multiplicative weight technique based on Littlestone and Warmuth's weighted majority algorithm [Littlestone and Warmuth, 1989], and use it to develop Adaboost.

More generally, regret minimization is intimately related to the well-studied area of universal prediction [Merhav and Feder, 1998], with its many related fields: universal coding in information theory, universal consistency in game theory, and online prediction in machine learning. Various authors have also showed that certain types of regret-minimization algorithms correspond to Bayes-optimal estimators in simple parameter estimation problems [Monteleoni and Jaakkola, 2003; Kakade and Ng, 2004].

2.2.2 A reactive opponent

It is important to note that most of the regret-minimization methods proposed by the machine learning community assume what is often called an *oblivious* opponent. That is, the opponent does not learn or react to our actions, and essentially plays a fixed string of actions. It is this assumption that makes it sensible to compare our performance against other possible choices of actions given the same history of opponent actions. Since the opponent is oblivious, we can assume that its history of actions is independent of our choices of actions.

Under most circumstances, we might expect an intelligent opponent to change their strategy as they observe our own sequence of plays. Thus, the opponent (and hence the environment) may be changing as we play. We can still use the algorithms described above, but we must simply satisfy ourselves with achieving a regret-bound relative to the best we could have done against the worst-case oblivious opponent.

On the one hand, this does not seem too bad. We will do as well as the best fixed action against the worst-case opponent. However, in many cases we do not face the worst-case opponent. We might actually be able to do better than what these worst-case algorithms guarantee. One possibility would be to consider opponents that play behavioral strategies that depend on our prior actions [Chang and Kaelbling, 2002].

For example, consider the game of Prisoner's Dilemna. If we follow the oblivious opponent assumption, then the best choice of action would always be to "Defect." However, all of the methods above would thus miss out on the chance to cooperate with opponents such as a "Tit-for-Tat" opponent. These opponents can be called *reactive* opponents. Mannor and Shimkin [2001] propose a super-game framework for extending the regret-minimization framework to handle such cases. In the super-game framework, we evaluate performance not based on single periods of play; instead each time we play an action or strategy, we commit to playing it for multiple time steps in order to allow the environment to react to our strategy.

Pucci de Farias and Megiddo [2004] propose a different algorithm and use a different performance metric for playing in such cases. Their performance metric measures the algorithm's performance against the best minimum average reward attained by any action or strategy within a certain time period. That is, it follows the same spirit as the regret-minimization methods by comparing performance against the best expert. However, in their case, each expert's performance is defined differently as the smallest average reward that the expert achieves within any phase in the history of plays, where each phase is a certain time interval of the history. This is somewhat unnatural, since it leads to a looser definition of good performance. If an algorithm is attains the reward of the best expert under this metric, it is only doing as well as the expert with the best poorest-performing phase. Thus, as long as all experts perform badly just some of the time, an algorithm designed to fare well under this metric does not actually have to perform very well at all. We will focus on the usual definition of performance in this thesis when we discuss regret-minimization methods.

Finally, Schmidhuber [1999] proposes a method he calls EIRA, Environment-Independent Reinforcement Acceleration. EIRA attempts to continually improve an agent's policy through modifications that improve the performance. If a policy begins to perform poorly, whether because of recent modification or because the environment has changed, EIRA switches to using the previous best strategy. Thus, EIRA guarantees that we are usually playing the best strategy we have evaluated, but again, regret-minimization is more powerful since it guarantees that we are achieving performance equivalent to the expected performance of the best policy over all periods of play.

2.3 Filtering

In the regret minimization framework discussed above, we assumed we did not know a model of the environment. That is, we could not predict future opponent actions based on observed past actions. However, sometimes we might have such a model, even if we assume that the opponent's strategy is changing over time. If we have a model of the way the opponent's strategy changes, then we can try to use our observations to deduce the true environmental/opponent state at each time period.

Kalman filtering is a simple and efficient method for doing this when our model of environmental change satisfies certain requirements. The celebrated and much-studied Kalman filter was originally developed as a way to track moving objects given noisy observations and a noisy model about the object's movement, or state change. The noise in both cases is assumed to be a Gaussian process, and the model for both observations and state change is linear.

To be precise, there is an additive noise process b_t that evolves according to $b_{t+1} = b_t + z_t$, where z_t is a zero-mean Gaussian random variable with variance σ_w. The global reward that it observes if it is in state i at time t is $g_t = r(i) + b_t$, where r is a vector containing the ideal training rewards $r(i)$ received by the agent at state i.

The standard model that describes such a linear system is:

$$g_t = Cx_t + v_t, \quad v_t \sim N(0, \Sigma_2)$$
$$x_t = Ax_{t-1} + w_t, \quad w_t \sim N(0, \Sigma_1)$$

where v_t and w_t are Gaussian noise, C and A are matrix operators, x_t is the state vector we wish to track, and g_t is a vector of observations that is available to us. In our case, $x_t = [rb]$, since we are trying to estimate the personal training reward signals and the noise term. Kalman filtering maintains an estimate of the state vector x_t and its full covariance matrix, updating these quantities as it gets new observations. We will discuss this model in more detail in Chapter 4.

Clearly, this model only describes a very particular set of environments that may happen to behave in this way. However, in Chapter 4, we will explore the applicability of this model to situations where the model conditions may not be entirely satisfied.

There have been many other techniques proposed to deal with environments that change over time. In slowly varying environments, Szita et al. [2002] provide a specialization of Littman and Szepesvári's [1996] techniques for generalized MDPs, showing that Q-learning will converge as long as the variation per time step is small enough. In our case, we will be attempting to tackle problems where the variation is much larger. Choi et al. [1999] investigate models in which there are "hidden modes". When the environment switches between modes, all the rewards may be altered. This works if we have fairly detailed domain knowledge about the types of modes we expect to encounter. For variation produced by the actions of other agents in the world, or for truly unobservable environmental changes, this technique would not work as well and regret-minimization methods may be better suited to handle such problems.

2.4 POMDP methods

More generally, we may have a complex model of the environment, in which we assume there is an underlying state space that is Markov. This underlying model might not change over time, but we might only receive observations of these underlying states, and the observations can be arbitrary functions of the underlying states. Thus, observations for different states may actually appear the same. This situation can be modeled as a *partially observable Markov decision process*, or POMDP. Theoretically, anything could be modeled this way given enough underlying states, but the computational challenges for deriving an optimal policy of behavior in such a model are formidable.

Formally, a POMDP is defined by:

- $|S|$ states $S = \{1, ..., |S|\}$ of the world (or environment),
- $|A|$ actions $A = \{1, ..., |A|\}$ that can be executed by the agent,
- a transition matrix P that describes the probability of transitioning from state i to state j given action a,
- observations O, and
- a reward function $r(i) \in \mathbb{R}$ for each state $i \in S$.

Since we are concerned with multi-agent settings, we note that the states S may represent both states of the world, or states of the other agents. Thus, more generally, we will refer to S as being the state space of the *environment*, which includes both the world and the other agents. Sometimes our observations will also include the actions of the other agents in the environment.

It is apparent that the POMDP framework is powerful enough to model many different types of opponents. As long as the opponent can be modeled by a finite number of states, this model is sufficient. If we expect that the opponent operates with a continuum of states or an infinite number of states, then we would need further extensions of the POMDP framework to continuous (and possibly infinite) state spaces, which we will not explore in this thesis. Given these limitations, we note that we cannot model opponents that employ arbitrary learning algorithms.

We must assume that there is some underlying MDP that describes the opponent's internal decision making mechanism, i.e. the opponent is assumed to be a probabilistic finite automaton. Unlike in the game theoretic setting, we are making a different type of assumption about the opponent. Rather than assuming the opponent is rational, we now assume that the opponent can be modeled by a hidden MDP. At the minimum, we only get to receive observations that are the opponents' actions at each time period.

In some sense, in the methods we have described so far, we could make three different assumptions about the opponent:

- rational: The concept of rationality is the basis for Nash equilibrium.
- worst-case: This leads to the idea of using minimax optimality.
- MDP: In this case, we can attempt to solve for the optimal policy if we can estimate the parameters of the MDP that describes the opponent.

2.4.1 Markov decision processes

Markov decision processes (MDPs) are significantly simpler than POMDPs since they assume direct observation of the world state. Thus, there is no need for the set of observation O, since it is equivalent to the states S.

An MDP is thus:

- $|S|$ states $S = \{1, ..., |S|\}$ of the world,
- $|A|$ actions $A = \{1, ..., |A|\}$ that can be executed by the agent,
- a transition matrix P that describes the probability of transitioning from state i to state j given action a,
- a reward function $r(i) \in \mathbb{R}$ for each state $i \in S$.

We are usually trying to learn a policy $\mu : S \rightarrow A$ that maps observed states to action choices. The goal is to learn a policy that provides us with the highest possible long-term rewards. We can define long-term reward in one of two ways:

Definition 5 *The long-term average reward, starting in state i and executing policy μ, and assuming the reward function $r(i)$ is bounded, is defined as*

$$\eta(\mu, i) = \lim_{T\rightarrow\infty} \frac{1}{T} E[\sum_{t=0}^{T} r(i_t)|i_0 = i, \mu].$$

Definition 6 *The long-term discounted sum of rewards is defined as*

$$J(\mu, i) = E[\sum_{t=0}^{\infty} \gamma r(i_t) | i_0 = i, \mu],$$

where $\gamma \in [0, 1)$ is the discount factor, and i_0 is the initial state of the agent, and i_t is the state of the agent at each subsequent time period t.

Conveniently, if the agent plays mixed policies and the POMDP becomes *ergodic*, then the two notions of reward are closely related to each other [Aberdeen, 2002; Baxter and Bartlett, 2001]. Let E_i be the expectation over the unique stationary distribution over states generated by such a mixed policy in the POMDP. Then

$$E_i J(\mu, i) = \frac{1}{1-\gamma} E_i \eta(\mu, i).$$

Given a complete description of the model (S, P, A, R), we can use policy iteration [Bertsekas and Tsitsiklis, 1996] or value iteration [Bertsekas and Tsitsiklis, 1996] to solve for the optimal policy.

If we do not know the model parameters, in this case P, and we only know S, we must either 1) learn the parameters and then solve for the optimal policy, or 2) learn the optimal policy directly without first learning the model parameters. The first method is usually referred to as a model-based method, since it relies on building a complete description of a MDP and solving this model for an optimal policy. The second method is often called a model-free method since it tries to learn an optimal policy without estimating the model's transition matrix. Both types of methods are the subject of much study in reinforcement learning. A good overview of these methods can be found in the survey article [Kaelbling *et al.*, 1996] or the textbook [Sutton and Barto, 1999].

2.5 Mobile ad-hoc networking

Finally, we discuss some prior work in the field of mobile ad-hoc networking, since one chapter of this thesis will be devoted to applying our algorithms to the closely related domain of *mobilized* ad-hoc networking, which we introduce in Chapter 6. We distinguish our application domain by calling it *mobilized* ad-hoc networking because we assume that our nodes have control over their own movements, whereas researchers who study mobile ad-hoc networks often assume that there are external users that force the nodes to move along with them. For example, cell phones are forced to move along in their owner's pockets or purses. In the mobilized ad-hoc networks that we study, we assume that nodes can be mobile robots that can determine their own preferred movements.

2.5.1 Node movement

We draw inspiration for this work from two different fields: networking and reinforcement learning. In the networking literature, some work on the effect of node mobility in ad-hoc networks has been done for applications in which movement and topology changes are on the time-scale of packet delivery. Nodes are then able to act as mobile relays physically transporting packets from one location to another. Grossglauser and Tse [2001] analyze a strategy in which source nodes send packets to as many different nodes as possible, which store the packets and hand them off whenever they get close to the intended destination nodes. Li and Rus [2000] consider a scenario in which mobile hosts make deviations from predetermined trajectories to transmit messages in disconnected networks. Chatzigiannakis et al [2001] consider the case where a subset of mobile nodes are constrained to move to support the needs of the protocol, and act as a mobile pool for message delivery.

2.5.2 Packet routing

Our work is in a different setting, in which topology changes are on a much longer time scale than packet delivery delay constraints. Nodes move in order to form and maintain connected routes, rather than to physically deliver packets in a disconnected network. Routing is thus an important aspect of our algorithms that influences and is informed by movement decisions. Perkins and Royer [1997] and Johnson and Maltz [1996] examine routing in mobile ad-hoc networks where no control over node movements is assumed, while Chen et al. [2001] deal with the issue of preserving a connected topology with only some portion of nodes awake at any one time.

2.5.3 Learning

From the reinforcement learning community, there has been some interest in applying learning techniques to improve network performance. Such adaptive algorithms may be better able to perform well under widely varying conditions. Boyan and Littman [1994] applied reinforcement learning techniques to the problem of routing in static networks. They showed that a simple adaptive algorithm based on the Q-learning algorithm [Watkins, 1989] can out-perform a shortest-paths algorithm under changing load and connectivity conditions. Peshkin [2002] used policy search rather than Q-learning on the same problem, which allowed the system to search a richer policy space. By using stochastic routing policies, the system is able to manage high loads by finding multiple possible source-destination paths. We apply Q-learning to the case of mobile networks, where node connectivity is constantly changing.

Moreover, since we assume control over the nodes' movements, we can also influence these connectivity changes by learning a good control mechanism for the node movements. Several papers mentioned above propose various methods for controlling node and packet movement under specific assumptions. In the general setting, we wish to select optimal actions at each time step to maximize the long-term system performance. This type of problem lends itself to reinforcement learning techniques [Kaelbling *et al.*, 1996], where the goal of the learner is to maximize long-term reward by learning an optimal behavior policy through simulation. Stone and Sutton [2001] and Bowling and Veloso [2002a] studied methods for scaling up reinforcement learning techniques to complex domains such as robotic soccer.

However, a major problem in the networking domain is the high degree of partial observability. From the perspective of any one node in the network, most of the rest of the network state cannot be discerned. The node can only rely on messages passed to it from its neighbors to glean information about the rest of the network, and to compound the issue, we would like to minimize such informational messages since they only contribute to network congestion. Thus, we will have to deal with partial observability in creative ways.

There are several other challenges that we face. For example, as we train the nodes, the behavior of the overall network changes as the nodes learn to use better policies. As the nodes move, the learning environment changes. These changes are further exacerbated by the fact that the other nodes are learning new behaviors over time as well.

Chapter 3

Learning in repeated games

We begin by investigating basic multi-agent learning principles in the simplest setting possible: repeated normal-form games. In this setting, all the complexity of the environment has been reduced to a simple matrix of rewards. The repeated matrix game is a degenerate form of a general stochastic games, in which there is only a single state. Thus, we no longer have to concern ourselves with estimating state transitions. All the remaining complexity becomes encapsulated in the behavior of our opponent. This is the main object of our study.

When we study an agent's behavior, we will often refer to its *policy* or *strategy*, which is the method by which it chooses an action to execute. We will use the terms policy and strategy interchangeably. As discussed earlier, these strategies can be stationary distributions over actions, or they can be more complex mappings from histories of observations to action choices. They might even be more complex, possibly assuming the form of some rule governing an agent's behavior based on various attributes of the environment and the history of play. In this sense, a strategy could in fact be any arbitrary algorithm. The algorithm receives inputs that are observations of the past periods of play, and outputs future action choices. Thus, we can consider the learning algorithms that we design for playing multi-agent games to be strategies themselves. An agent's choice of a particular learning algorithm at the beginning of a game would amount to a choice of a particular strategy that it will follow for the rest of the game.

Table 3.1: Summary of multi-agent learning algorithms under our new classification that categorizes algorithms by their observation history length and belief space complexity.

	$\mathcal{B}_0$	$\mathcal{B}_1$	$\mathcal{B}_X$	$\mathcal{B}_t$	$\mathcal{B}_\infty$
$\mathcal{H}_0$	minimax-Q, Nash-Q				Bully
$\mathcal{H}_1$					Godfather
$\mathcal{H}_t$					
$\mathcal{H}_\infty$	Q-learning (Q_0), (WoLF-)PHC, fictitious play, Exp3	Q_1	PHC-Exploiter	Hedged learning	

3.1 A new classification

We propose a general classification that categorizes multi-agent learning algorithms by the cross-product of their possible strategies and their possible beliefs about the opponent's strategy, $\mathcal{H} \times \mathcal{B}$. An agent's possible strategies can be classified based upon the amount of history it has in memory, from $\mathcal{H}_0$ to $\mathcal{H}_\infty$. Given more memory, the agent can formulate more complex policies, since policies are maps from histories to action distributions.

At the simplest end of the spectrum, $\mathcal{H}_0$ agents are memoryless and can only play stationary policies. They cannot condition their action choices on their observation of the past sequences of plays; i.e., they only play constant policies $\mu : 1 \rightarrow PD(A)$. For example, an agent that always plays $(1/3, 1/3, 1/3)$, i.e. a stationary distribution that equally weights each possible action, in the game of Rock-Papers-Scissors would be an $\mathcal{H}_0$ agent.

Agents that can recall the actions from the previous time period are classified as $\mathcal{H}_1$ and can execute purely reactive policies. These agents can possess policies that map the last period's observed action choices to a distribution over action choices for the current period, $\mu : h_{t-1} \rightarrow PD(A)$. This can be extended to agents with a finite amount of memory, who react to sequences of plays from within the last k time periods. These $\mathcal{H}_k$ agents would be able to execute fixed behavioral strategies of the form $\mu : H_k \rightarrow PD(A)$, where $H_k = \bigcup_{\tau=1}^{k} H^k$ is the set of all possible histories of length k or less. Each strategy would thus be a complete mapping from any observed history of length k or less to a particular distribution over action choices.

At the other extreme, $\mathcal{H}_\infty$ agents have unbounded memory and can formulate ever more complex strategies as the game is played over time. Most adaptive, learning algorithms will fall under this category, since these algorithms make use of the observed history of play in order to improve their learned policies.

An agent's belief classification mirrors the strategy classification in the obvious way. Agents that believe their opponent is memoryless are classified as $\mathcal{B}_0$ players, $\mathcal{B}_t$ players believe that the opponent bases its strategy on the previous t-periods of play, and so forth. Although not explicitly stated, most existing algorithms make assumptions about the types of possible opponents in the world. These assumptions are embodied in the kinds of beliefs they hold about the opponent.

For example, consider a $\mathcal{B}_0$ player. This player assumes that the opponent is essentially playing a fixed mixed policy, i.e. the opponent plays according to a stationary distribution over its possible actions, $\mu^{-i} : \{1\} \rightarrow PD(A)$. Clearly, given enough observations of such an opponent's past history of play, we can estimate the stationary distribution fairly accurately. Thus, if the opponent truly is playing according to a stationary distribution, a $\mathcal{H}_t \times \mathcal{B}_0$ or $\mathcal{H}_\infty \times \mathcal{B}_0$, with t sufficiently large, ought to be able to perform fairly well.

Of course, a player with beliefs that can be categorized as $\mathcal{B}_t$ clearly subsumes a $\mathcal{B}_0$ player, in the sense that a $\mathcal{B}_0$ player is simply a special case of a $\mathcal{B}_t$ player, with $t = 0$. A $\mathcal{B}_t$ player assumes that its opponents play according to fixed behavioral strategies, i.e. they play according to a policy that maps finite histories of observations to action choices, $\mu^{-i} : H_t \rightarrow PD(A)$. In other words, it assumes that its opponent is a $\mathcal{H}_t$ player. A $\mathcal{B}_0$ player is thus simply a degenerate $\mathcal{B}_t$ player that assumes that the opponent maps all possible observation histories H_t to the same probability distribution over actions $PD(A)$.

We also introduce the notion of a $\mathcal{B}_X$ player, which does not assume that the opponent is capable of playing arbitrary policies that use the entire history observations, but which also does not limit the opponent to using the previous t periods of history in formulating its strategy. The $\mathcal{B}_X$ player assumes that the opponent uses some compressed version of the history of observations in deciding which strategy to play. For example, the opponent may keep track of our most common action over history, and play a best response to that action. Let this feature space that the opponent uses to model our play be called its feature history X. Then a $\mathcal{B}_X$ player assumes that it faces opponents which are capable of playing policies of the form $\mu^{-i} : X \rightarrow PD(A)$.

Finally, a $\mathcal{B}_\infty$ player makes the most general assumptions about its possible opponents. In fact, since it assumes that the opponent is a $\mathcal{H}_\infty$ player, it essentially makes no assumptions about the opponent. The opponent could be using an arbitrarily complex, adaptive algorithm. Clearly, it would be a great achievement to design a learning algorithm that performs optimally given such general assumptions. Unfortunately, as we discussed in Chapter 2, Nachbar and Zame [1996] have shown that this is impossible. Thus, even if we use all available history, we cannot expect to design an optimal $\mathcal{H}_\infty \times \mathcal{B}_\infty$ player, since the optimal strategy may not even be computable by a Turing machine. We will have to satisfy ourselves with either creating algorithms with simpler opponents assumptions (say, $\mathcal{B}_t$ or $\mathcal{B}_X$, for various spaces X) or algorithms that can perform reasonably, if not optimally, against all opponents. How we define "reasonable" is an important question, which we will address in much more detail in Chapter 5.

As we seek to design good multi-agent learning algorithms, we can think of each $\mathcal{H}_s \times \mathcal{B}_t$ as a different *league* of players, with players in each league roughly equal to one another in terms of their capabilities. Clearly some leagues contain less capable players than others. We can thus define a *fair* opponent as an opponent from an equal or lesser league. New learning algorithms should ideally be designed to beat any fair opponent.

Definition 7 *A fair opponent for a player in league $\mathcal{H}_s \times \mathcal{B}_t$ is any player from a league $\mathcal{H}_{s'} \times \mathcal{B}_{t'}$, where $s' \leq s$ and $t' \leq t$.*

3.1.1 The key role of beliefs

Within each league of players (algorithms), let us assume that we can create players are fully rational in the sense that they can fully use their available histories to construct their future policy. However, an important observation is that the definition of full rationality depends on their beliefs about the opponent. If we believe that our opponent is a memoryless player, then even if we are an $\mathcal{H}_\infty$ player, our fully rational strategy is to simply model the opponent's stationary strategy and learn to play our stationary best response. While our path of play may appear adaptive in the short run because we are estimating the opponent's stationary distribution as we play, in the long run our strategy is expected to converge to a steady state. When this player is faced with an opponent that does not play a stationary distribution every time period, then this player's strategy may never converge. Moreover, since it cannot accurately model the opponent due to the limitations of its belief space, we would not expect its performance to be very good. Given the entire history of observations, its best response given no further information would be to play a stationary distribution, even though more complex strategies might clearly perform better. Thus, our belief capacity and our history capacity are inter-related. Without a rich set of possible beliefs about our opponent, we cannot make good use of our available history. Similarly, and perhaps more obviously, without a rich set of historical observations, we cannot hope to model complex opponents.

3.1.2 Existing algorithms

Many of the existing algorithms fall within the $\mathcal{H}_\infty \times \mathcal{B}_0$ league. As discussed above, the problem with these players is that even though they have full access to the history, given no further observations of future play, their fully rational strategy is stationary due to their limited belief set. This is a reasonable first step in terms of designing competitive multi-agent algorithms; however, it is hardly sufficient in most practical circumstances. The players in this league are only adaptive in the sense that they continue to use all the observed history to refine their policy of play towards a hope-fully optimal stationary policy. They use the history to construct an estimate of the opponent's policy, which they assume to be fixed.

Such an algorithm essentially treats its environment as a Markov decision process, where the states in the MDP model correspond only to the external world states. Since the opponent is assumed to be fixed, we do not need to model its internal states. The actions in this MDP are simply the agent's own actions. The opponent's actions influence the transition probabilities between states in the state space, but since the opponent action distributions at each state are expected to be stationary, we can simply estimate the transition probabilities between the states without worrying further about the actual actions that the opponent takes. Thus, any standard reinforcement learning algorithm, using an assumption of a fixed opponent, would fall into this category of a $\mathcal{H}_\infty \times \mathcal{B}_0$ player, where the MDP model it tries to learn only depends on the external world states, its own actions, and the corresponding rewards that are observed.

Another general example of a $\mathcal{H}_\infty \times \mathcal{B}_0$ player is the policy hill climber (PHC). It maintains a policy and updates the policy based upon its history in an attempt to maximize its rewards. We will discuss this algorithm in more detail in the next section. It will be used as an example of an adaptive algorithm that can still be exploited by an intelligent opponent, because its beliefs about its possible opponents are very limited.

At the opposite end of the spectrum, Littman and Stone [2001] propose algorithms in $\mathcal{H}_0 \times \mathcal{B}_\infty$ and $\mathcal{H}_1 \times \mathcal{B}_\infty$ that are *leader* strategies in the sense that they choose a fixed strategy and hope that their opponent will "follow" by learning a best response to that fixed strategy. Their "Bully" algorithm chooses a fixed memoryless stationary policy, while "Godfather" has memory of the last time period. In games of multiple equilibria, these algorithms choose the equilibrium point most beneficial to themselves and fix their strategy at that equilibrium strategy, hoping the other agent will recognize that the best response is to play the other half of that equilibrium point. Opponents included normal Q-learning and Q_1 players, which are similar to Q-learners except that they explicitly learn using one period of memory because they believe that their opponent is also using memory to learn. The interesting result is that "Godfather" is able to achieve an equilibrium against Q_1 in the repeated prisoner's dilemna game that is not an equilibrium of the one-shot game, with rewards for both players that are higher than the one-shot Nash equilibrium rewards. This demonstrates the power of having belief models. Q_1 is a $\mathcal{B}_1$ player and Bully is a $\mathcal{B}_\infty$ player.

However, the Bully and Godfather algorithms still rely on the smartness of the opponent in order to achieve good results. Because these algorithms do not have access to more than one period of history, they cannot begin to attempt to construct statistical models of the opponent. "Godfather" works well because it has a built-in best response to Q_1 learners rather than attempting to learn a best response from experience.

Finally, Hu and Wellman's Nash-Q and Littman's minimax-Q are classified as $\mathcal{H}_0 \times \mathcal{B}_0$ players, because even though they attempt to learn the Nash equilibrium through experience, their play is fixed once this equilibrium has been learned. Furthermore, they assume that the opponent also plays a fixed stationary Nash equilibrium, which they hope is the other half of their own equilibrium strategy. Thus, if we assume that the joint rewards of the repeated matrix game or stochastic game are known before the game even begins, then Nash-Q and minimax-Q will not need to learn or adapt at all; they will simply solve for a Nash equilibrium of the given game and play according to it. Nash-Q, in particular, requires the assumption that an oracle is available that ensures that all the players in the game choose the same Nash equilibrium, since there may be multiple equilibria.

A summary of these different algorithms is given in Table 3.1. We will discuss Exp3 and Hedged Learning further in Chapter 5. Rather than claiming optimality against a particular opponent type, these regret-minimizing algorithms perform approximately optimally against some set of opponents, and perform approximately as well as some comparison class of strategies against any arbitrary opponent. This will turn out to be a very appropriate way of evaluating the performance of multi-agent learning algorithms, as we will discuss in Chapter 5. In the meantime, we will further explore the shortcomings of the other approaches we have classified in Table 3.1, such as the various $\mathcal{H}_\infty \times \mathcal{B}_0$ players.

3.1.3 The policy hill climber

A general example of a $\mathcal{H}_\infty \times \mathcal{B}_0$ player is the policy hill climber (PHC). It maintains a policy and updates the policy based upon its history in an attempt to maximize its rewards. Originally PHC was created for stochastic games, and thus each policy also depends on the current state s. $\mu(s)$ denotes the mixed policy the agent employs when it observes itself at state s. In a slight abuse of notation, $\mu(s,a)$ will denote the probability assigned to taking action a at state s by the mixed policy $\mu(s)$.

In the repeated games that we study in this chapter, there is only one state, so s is fixed at that one state. In later chapters, we will be dealing with stochastic games that possess multiple states, so we will preserve the notation of using a state s in the following description of the policy hill climbing algorithm.

For agent i, Policy Hill Climbing (PHC) proceeds as follows:

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

$$Q(s,a) \leftarrow 0, \mu_i(s,a) \leftarrow \frac{1}{|A_i|} \forall s \in S, a \in A_i.$$

2. Repeat,

a. From state s, select action a according to the mixed policy $\mu_i(s)$ with some exploration.

b. Observing reward r and next state s', update

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

c. Update $\mu(s,a)$ and renormalize it to a legal probability distribution:

$$\mu_i(s,a) \leftarrow \mu_i(s,a) + \begin{cases} \delta & \text{if } a = \operatorname{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}.$$

The basic idea of PHC is that the Q-values help us to define a gradient upon which we execute hill-climbing. As we learn, we always increase the probability of choosing the action with highest Q-value in an attempt to increase our expected future rewards. Normal Q-learning, if applied to stochastic games, would be a special case of PHC with $\delta = 1$, and Bowling and Veloso's WoLF-PHC (Win-or-Lose-Fast PHC) [Bowling and Veloso, 2002b] modifies PHC by adjusting δ depending on whether the agent is "winning" or "losing." True to their league, PHC players play well against stationary opponents. WoLF-PHC also manages to converge to a stationary Nash equilibrium against other PHC players.

3.2 A new class of players and a new algorithm

As discussed above, most existing algorithms do not form beliefs about the opponent beyond $\mathcal{B}_0$. None of these approaches is able to capture the essence of game-playing, which is a world of threats, deceits, and generally out-witting the opponent. We wish to open the door to such possibilities by designing learners that can model the opponent and use that information to achieve better rewards. Ideally we would like to design an algorithm in $\mathcal{H}_\infty \times \mathcal{B}_\infty$ that is able to win or come to an equilibrium against any fair opponent.

Since this is impossible [Nachbar and Zame, 1996], we start by proposing an algorithm in the league $\mathcal{H}_\infty \times \mathcal{B}_\infty$ that plays well against a restricted class of opponents. Since many of the current algorithms are best-response players, we choose an opponent class such as PHC, which is a good example of a best-response player in $\mathcal{H}_\infty \times \mathcal{B}_0$. In the remainder of this chapter, we will describe our new algorithm and demonstrate empirically that our algorithm indeed beats its PHC opponents and in fact does well against most of the existing fair opponents.

In Chapter 5, we will extend these ideas to create an algorithm that achieves good performance against a large set of possible opponents. This *hedged learning* algorithm provides theoretical guarantees about its performance against any opponent, and also performs reasonably well in practice.

3.2.1 A new algorithm: PHC-Exploiter

Our algorithm is different from most previous work in that we are explicitly modeling the opponent's learning algorithm and not simply his current policy. In particular, we would like to model players from $\mathcal{H}_\infty \times \mathcal{B}_0$. Since we are in $\mathcal{H}_\infty \times \mathcal{B}_\infty$, it is rational for us to construct such models because we believe that the opponent is playing strategies that depend on history, that is, we believe that the opponent is learning and adapting to us over time using its history.

The idea is that we will "fool" our opponent into thinking that we are stupid by playing a decoy policy for a number of time periods and then switch to a different policy that takes advantage of their best response to our decoy policy. From a learning perspective, the idea is that we adapt much faster than the opponent; in fact, when we switch away from our decoy policy, our adjustment to the new policy is immediate. This is in contrast to our opponent, which adjusts its policy by only a small increment after each time period. Moreover, due to the fact that the opponent is in $\mathcal{H}_\infty \times \mathcal{B}_0$, its assumptions about us render it unable to model our changing behavior due to its design assumptions. We can repeat this "bait and switch" cycle ad infinitum, thereby achieving infinite total rewards as $t \to \infty$. The opponent never catches on to us because it believes that we only play stationary policies.

A good example of such a $\mathcal{H}_\infty \times \mathcal{B}_0$ player is Policy Hill Climber, with its variants WoLF-PHC and Q-learning. Iterated Gradient Ascent (IGA) is a special case of PHC for two-player, two-action games with public knowledge of the opponent's policy. Public knowledge refers to the fact that we can observe the probability distribution with which the opponent chooses an action to play. This is in contrast to the usual situation where we only get to observe the action actually chosen.

Bowling and Veloso showed that in self-play, a restricted version of WoLF-PHC always reaches a stationary Nash equilibrium in two-player two-action games, and that the general WoLF-PHC seems to do the same in experimental trials involving more general games. Thus, in the long run, a WoLF-PHC player achieves its stationary Nash equilibrium payoff against any other PHC player. In cooperative settings, this is a desirable outcome since both players are doing well over the long-run. The interesting case is in the competitive setting, where one agent's gain is generally the opponent's loss. In particular, the Nash equilibrium payoff for both players of a purely competitive, zero-sum game is zero.

We wish to do better than that by exploiting our knowledge of the PHC opponent's learning strategy. We can construct a *PHC-Exploiter* algorithm for agent i that proceeds like PHC in steps 1-2b, and then continues as follows:

1. Let α be the learning rate, and w be the window of estimation. Initialize

$$Q(s,a) \leftarrow 0,\ \mu_i(s,a) \leftarrow \frac{1}{|A_i|},\ \ \forall s \in S, a \in A_i, \qquad \delta_i \leftarrow 0 \ \ .$$

2. Repeat,

a,b. Same as PHC.

c. Observing action a_{-i}^t at time t, update our history h and calculate an estimate of the opponent's policy:

$$\hat{\mu}_{-i}^t(s,a) = \frac{n_{t,s,a}}{n_{t,s}} \quad \text{for all } a \ \ ,$$

where $n_{t,s,a}$ is the number of times in periods $t-w$ to t that the opponent played action a at state s, and $n_{t,s}$ is the number of times in periods $t-w$ to t that the opponent faced state s. We denote the length of the window of estimation as w. We estimate $\hat{\mu}_{-i}^{t-w}(s)$ similarly.

d. Update δ by estimating the learning rate of the PHC opponent:

$$\delta \leftarrow \frac{\max_a \left|\hat{\mu}_{-i}^t(s,a) - \hat{\mu}_{-i}^{t-w}(s,a)\right|}{w} \ \ .$$

e. Update $\mu_i(s,a)$. If we are winning, i.e., $\sum_{a'} \mu_i(s,a')Q(s,a') > R_i(\hat{\mu}_i^*(s), \hat{\mu}_{-i}(s))$, then update

$$\mu_i(s,a) \leftarrow \begin{cases} 1 & \text{if } a = \operatorname{argmax}_{a'} Q(s,a') \\ 0 & \text{otherwise} \end{cases} \ \ ;$$

otherwise, we are losing, then update

$$\mu_i(s,a) \leftarrow \mu_i(s,a) + \begin{cases} \delta & \text{if } a = \operatorname{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}.$$

Note that we derive both the opponent's learning rate δ and the opponent's policy $\hat{\mu}_{-i}(s)$ from estimates using the observable history of actions. If we assume the game matrix is public information, then we can solve for a Nash equilibrium strategy $\hat{\mu}_i^*(s)$, otherwise we can run WoLF-PHC for some finite number of time periods to obtain an estimate of this equilibrium strategy. Similar to the assumptions made in the WoLF-PHC algorithm, there is assumed to be only Nash equilibrium in the games considered here. The main idea of this algorithm is that we take full advantage of all time periods in which we are winning and getting higher payoffs than our Nash equilibrium payoffs, that is, when $\sum_{a'} \mu_i(s,a')Q(s,a') > R_i(\hat{\mu}_i^*(s), \hat{\mu}_{-i}(s))$.

It takes the PHC opponent a significant amount of time to realize that it is being beaten and to adjust its policy appropriately, and during this time, we reap significant rewards. Once we begin losing, we can then lead the opponent to a new policy that sets it up for another beating. It is willing to go along because it is optimizing its current policy by increasing the probability of choosing the action with highest Q-value. Because the Q-value essentially absorbs some randomness of the play depending on the update rule, we do not have to lose in every time period. By the time we have the opponent set up for another beating, we will have lost much less reward than we gained during the period of advantage.

3.2.2 Analysis

The PHC-Exploiter algorithm is based upon PHC and thus exhibits the same behavior as PHC in games with a single pure Nash equilibrium. Both agents generally converge to the single pure equilibrium point. The interesting case arises in competitive games where the only equilibria require mixed strategies, as discussed by Singh et al. [2000] and Bowling and Veloso [2002b]. Matching pennies, shown in Figure 1(a), is one such game. In this type of game, PHC-Exploiter is able to use its model of the opponent's learning algorithm to choose better actions for itself. In the long-run, it achieves better rewards than any PHC opponent.

In the full knowledge case, where we know our opponent's policy μ_2 and learning rate δ_2 at every time period, we can prove that a PHC-Exploiter learning algorithm will guarantee us unbounded reward in the long run playing games such as matching pennies. The central idea is that play will keep cycling, alternating between stages where we are gaining high reward and stages where we are losing some reward but setting up the system for another stage of high gain.

Proposition 8 *In the zero-sum game of matching pennies, where the only Nash equilibrium requires the use of mixed strategies, PHC-Exploiter is able to achieve unbounded rewards as $t \to \infty$ against any PHC opponent given that play follows the cycle C defined by the arrowed segments shown in Figure 2.*

Play proceeds along C_w, C_l, then jumps from (0.5, 0) to (1,0), follows the line segments to (0.5, 1), then jumps back to (0, 1). Given a point $(x, y) = (\mu_1(H), \mu_2(H))$ on the graph in Figure 2, where $\mu_i(H)$ is the probability by which player i plays Heads, we know that our expected reward is

$$R_1(x, y) = 1 - 2xy.$$

We wish to show that

$$\int_C R_1(x,y)dt = 2 \times \left(\int_{C_w} R_1(x,y)dt + \int_{C_l} R_1(x,y)dt \right) > 0 \quad .$$

We consider each part separately. In the losing section, we let $g(t) = x = t$ and $h(t) = y = 1/2 - t$, where $0 \le t \le 1/2$. Then

$$\int_{C_l} R_1(x,y)dt = \int_0^{1/2} R_1(g(t), h(t))dt = -\frac{1}{12} \quad .$$

Similarly, we can show that we receive 1/4 reward over C_w. Thus, $\int_C R_1(x,y)dt =$

$1/3 > 0$, and we have shown that we receive a payoff greater than the Nash equilibrium payoff of zero over every cycle. It is easy to see that play will indeed follow the cycle C to a good approximation, depending on the size of δ_2.

At time t, the policy of a PHC-Exploiter agent 1 playing against a PHC agent 2 is determined by the tuple (μ_1, μ_2, δ_2) at time $t-1$. For $\mu_1(H) > 0.5$ and $\mu_2(H) \leq 0.5$, agent 1 plays H exclusively. Agent 2 is adjusting its policy by δ_2 increments. Thus, we are guaranteed to reach $(1, 0.5)$. For $\mu_1(H) > 0.5$ and $\mu_2(H) \geq 0.5$, agent 1 is losing and adjusts its policy by δ_2 at each time step. By definition, the PHC agent 2 is also adjusting its policy by δ_2 increments. Thus, if we begin at $(1, 0.5)$, we will eventually reach $(0.5, 1)$. Continuing in this fashion, it is easy to see that the resulting actions follow the cycle depicted in Figure 2. The graph depicts the probability that each agent plays heads (H). In the vertical sections of the cycle C_w, agent 1 is winning; and in the diagonal sections C_l, agent 1 is losing and adjusts its policy by δ_2 increments. Note that the two winning sections are symmetric, as are the two losing sections. Thus, it remains to show that during a complete traversal of this cycle of policies, agent 1 receives positive total reward.

3.2.3 Experimental results

We now demonstrate empirically that we can estimate μ_2 and δ_2 sufficiently well from past observations, thus eliminating the full knowledge requirements that were used to ensure the cyclic nature of play above. We used the PHC-Exploiter algorithm described above to play against several PHC variants in different iterated matrix games, including matching pennies, prisoner's dilemna, and rock-paper-scissors. Here we give the results for the matching pennies game analyzed above, playing against WoLF-PHC. We used a window of $w = 5000$ time periods to estimate the opponent's current policy μ_2 and the opponent's learning rate δ_2. As shown in Figure 2, the play exhibits the cyclic nature that we predicted. The two solid vertical lines indicate periods in which our PHC-Exploiter player is winning, and the dashed, roughly diagonal, lines indicate periods in which it is losing.

In the analysis given above, we derived an upper bound for our total rewards over time, which was $1/6$ for each time step. Since we have to estimate various parameters in our experimental run, we do not achieve this level of reward. We gain an average of 0.08 total reward for each time period. Figure 3 plots the total reward for our PHC-Exploiter agent over time. In this experiment, we held the learning rate for the PHC player constant. The periods of winning and losing are very clear from this graph. If we let the learning rate for the PHC player decay as usual, the periods we observe will grow longer and longer over time.

Further experiments tested the effectiveness of PHC-Exploiter against other fair opponents, including itself. Against all the existing fair opponents shown in Table 3.1, it achieved at least its average equilibrium payoff in the long-run. This is shown in Table 3.2. Not surprisingly, it also posted this score when it played against a multiplicative-weight learner. We included experiments using the Matching Pennies game, the Rock-Papers-Scissors game, and Prisoner's Dilemma, which can be found in Figure 2-1. Each matchup of different players in each game was run for 20 trials, and the performance averaged over 500,000 periods. Matching Pennies and Rock-Papers-Scissors are both zero-sum games, while Prisoner's Dilemma is a general, non-zero-sum game. We see that in the zero sum games, the PHC-Exploiter is always able to beat or approximate its minimax payoff of zero, against all of its opponents. The Prisoner's Dilemma game is in some sense an easy game, since all the algorithms will quickly converge on the single-shot Nash equilibrium strategy and receive payoff of one in each time period, with all players defecting every round.

3.3 Combinations of methods

As we saw in the previous section, once an opponent gains some knowledge about our policies and learning algorithms, they can easily exploit our weaknesses. In response, we have a couple options. One method would try to hide our policies and algorithms from the opponent by purposefully obfuscating them. This obfuscation might incur us some loss, but the goal would be to minimize this loss while maximizing the obfuscating effect.

Game type	Opponent	Average reward	Standard Deviation
Matching pennies	PHC	0.081	0.0002
Matching pennies	MW	0.004	0.0011
Matching pennies	Nash-Q	-0.001	0.0005
Matching pennies	PHC-Exploiter	0.001	0.0008
Rock-Papers-Scissors	PHC	0.012	0.0044
Rock-Papers-Scissors	MW	-0.006	0.0429
Rock-Papers-Scissors	Nash-Q	0.018	0.0072
Rock-Papers-Scissors	PHC-Exploiter	0.008	0.0191
Prisoner's Dilemma	PHC	1.0	0.0013
Prisoner's Dilemma	MW	1.0	0.0021
Prisoner's Dilemma	Nash-Q	1.0	0.0009
Prisoner's Dilemma	PHC-Exploiter	1.0	0.0007

Table 3.2: Table showing performance of the PHC-exploiter against different opponents in various repeated games. Performance is given as average rewards per time period, over 20 trials. The game payoffs are given in Figure 2-1.

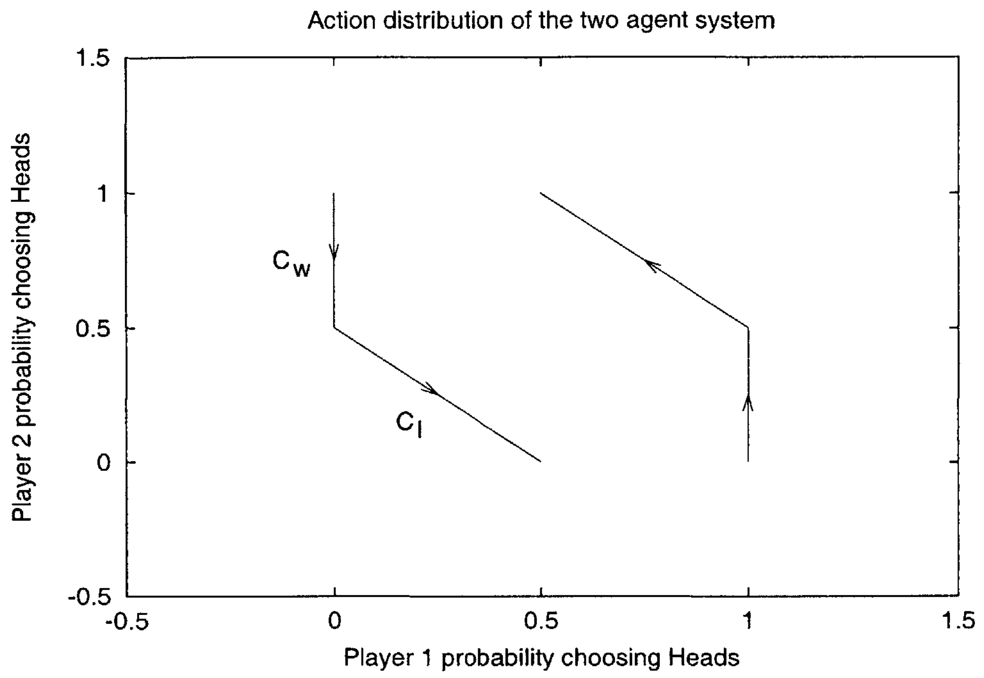
A second option would involve trying to identify the opponent's new, more complicated, policies and algorithms, and find a good response strategy to this more sophisticated opponent. With each advance in sophistication, our space of possible opponent models grows. In this way, it is similar to an arms race between two adversaries. We each need to increase our range of weapons, but it will be a never-ending cycle of increases. Nevertheless, short of negotiating a compromise solution, possibly an equilibrium solution, this is the situation we face. Opponents will always try to exploit possible weaknesses, and we must always be ready to defend ourselves against such exploits.

With an ever-increasing array of possible opponents, one major problem for us is opponent identification. Once we identify the opponent, then we can employ our best-response strategy to that type of opponent.

However, there is potential for skipping a step in this process. Similar to the difference between model-free learning and model-based learning, we can attempt to learn a good policy without identifying the opponent first. This is similar to learning an optimal policy for acting in an MDP without first learning the MDP parameters.

One way to achieve this is by mixing between our best-response policies for different types of opponents. If a policy seems to be working well, we play it more often. This intuitive framework is formally described by the mixture of experts approach, which provides us with bounds on the regret that we might suffer following such an approach. We will discuss this further in Chapter 5.

3.4 Conclusion



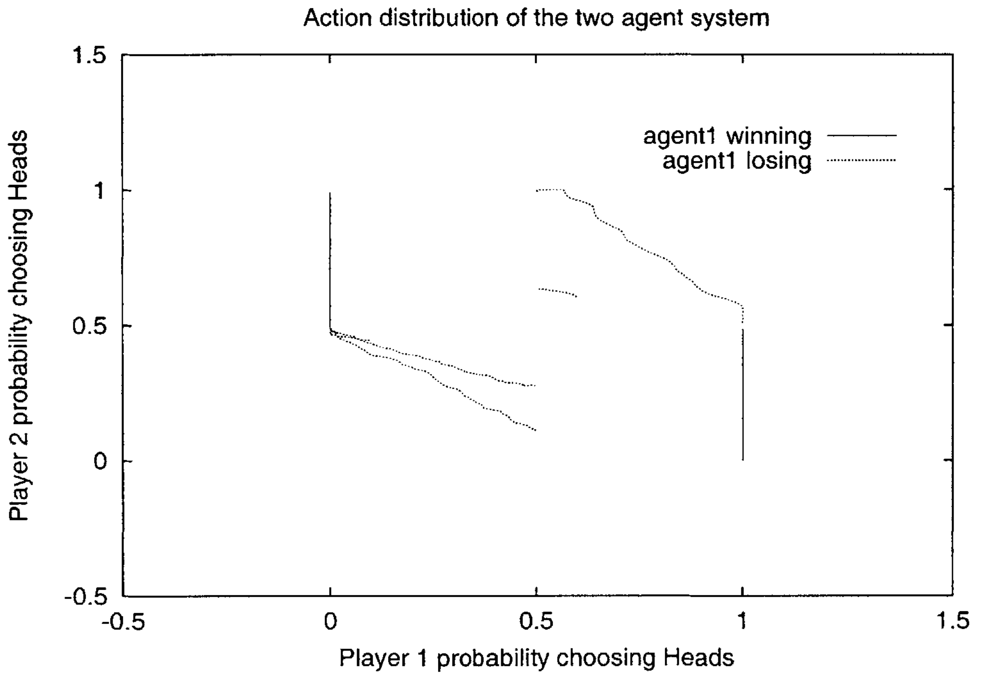


Figure 3-1: Theoretical (top), Empirical (bottom). The cyclic play is evident in our empirical results, where we play a PHC-Exploiter player 1 against a PHC player 2 in the Matching Pennies game.

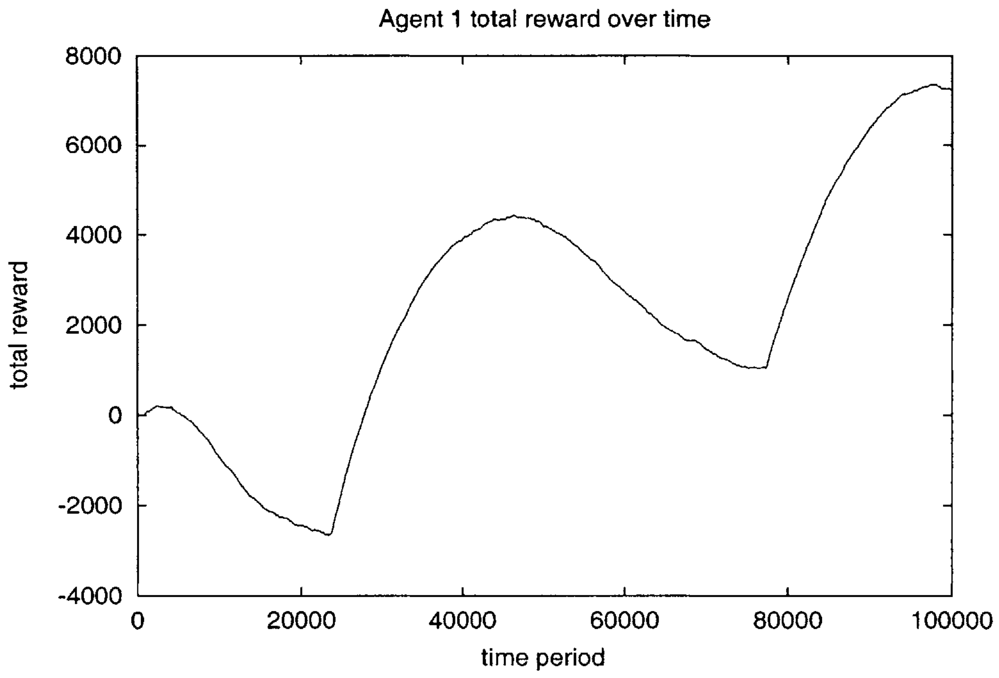


Figure 3-2: Total rewards for the PHC-Exploiter increase as we gain reward through each cycle, playing against a PHC opponent in the Matching Pennies game.

Chapter 4

Large sequential games

In the previous chapter, we focused on simple repeated matrix games, usually with a small number of players or participants. This allowed us to gain insight into the complex relationships between the different players' possible policies and beliefs. However, many real-world situations would be better modeled by a richer representation that included different states. Recall that in the repeated matrix game model, there is only one state of the world which we returned to at every time period. We thus play the same matrix game at every time period.

4.1 The problem setting

In this chapter, we will investigate richer representations that allow us to reason about different states of the world if necessary. For example, in a game of chess, the different states might correspond to the different board configurations. We still need to reason about the opponent's internal mental state, but we also have the additional complication of the external world state.

4.1.1 Stochastic games

A more general representation is the stochastic game, which extends the repeated game model by allowing states. Each state corresponds to a different matrix game, and the joint actions of the players at one state influence the probability with which they transition to a new state of the world. Whereas in the previous chapter, we only had to concern ourselves with the internal state of the opponent, here we also need to worry about the external state of the world in which we find ourselves. Not only will we want to manipulate the opponent so that it is in a "good state of mind", we also need to ensure that we both take actions that allow us to spend as much time as possible in good states of the world.

We might also refer to these stochastic games as sequential games, since past actions influence our current world state. However, this is an overloaded term. In game theory, sequential games usually refer to situations where the players can take actions sequentially, rather than simultaneously, as we have assumed. We will continue to use the simultaneous action assumption, since the sequential version of these games is actually often easier to deal with. For example, a sequential version of Rock-Papers-Scissors would be trivial for the second player, and always a losing proposition for the first player.

4.1.2 Partial observability

To further complicate matters, in many real-world domains, a single agent is unable to observe the overall state of the world, or the actions which all the other agents in the world take at each time period. In these large multi-agent domains, we are left with a partially observable external environment. In the previous section, the only partial observability stemmed from our inability to read an opponent's mind. We assumed we could always observe the opponents' actions, and clearly we also always knew the state of the world, since there was only a single state.

In the scenarios we consider in this chapter, our problems are compounded. Not only are the opponents' internal states of mind unobservable, the environment has become so complex that we may not be able to observe the entire environment at once. For example, the state space may be so large that we can only observe some local properties of the environment. If there are many agents operating in this large environment, perhaps we can only observe the locations of those agents which are close by.

An additional complexity arises from the use of the stochastic games model rather than simple repeated normal-form games. In stochastic games, we need to worry about the effect of our current actions on our trajectory of future states, and thus, future rewards. In repeated games, the future state is the same at every period, since we play the same normal-form game each time period, regardless of our chosen action. In stochastic games, depending on which action we choose in this current time period, we may transition into a different state, and thus play a different matrix game that corresponds to that different state during the next time period.

While this relationship of causality also existed in the repeated matrix games, it only existed in our model of the opponents' internal state. Our actions might affect its choice of actions in the future because we have either revealed something about ourselves, caused it to behave in a friendly way towards us, or otherwise affected its behavior due to its observation of our current actions. This causality was only implicit, since we could not actually observe the internal change of state in the opponents' minds; we could only observe the external outcomes of its effect on their resultant behaviors, if any. In stochastic games, on the other hand, the causality between current actions and future states is more explicit, since we may actually be able to observe the next state directly.

4.1.3 A motivating application

Stochastic games are relevant for studying a wide range of real world multiagent phenomenon. While much of the work in this thesis can be applied to many general domains, in this chapter we will restrict our study to fully cooperative situations where a large number of agents must interact and try to achieve the high level of global reward. This type of scenario is in part motivated by our interest in mobilized ad-hoc networks, which are described are studied in greater detail in Chapter 6. In short, achieving maximum performance in a mobilized ad-hoc network requires the connection of as many sources nodes as possible to the one or more receiver nodes in the system. We use learning techniques to train mobile agent nodes to move around and form connections between the sources and receivers, which may also be moving. During this training phase, we use the network connectivity as a measure of system performance, and pass that along to the mobile agent nodes as a global reward or training signal.

There are many further complications to learning in this mobilized ad-hoc networking setting. In this domain, the learning agent does not have a full view of the world – it cannot see the world state of agents that are far away or otherwise obscured. Furthermore, it certainly does not have a complete representation of the internal states of the other agents. This partial observability creates problems when the agent begins to learn about the world, since it cannot see how the other agents are manipulating the environment and thus it cannot ascertain the true world state. Aliasing may occur, a situation in which the agent cannot distinguish between one good state and one bad state because the observations it receives in both states are the same. This is an acute problem in partially-observed multi-agent learning, since the unobserved agents could have a tremendous effect on the value of a state.

4.1.4 Credit assignment

A separate problem arises when we train multiple agents using a global reward signal. Even with full observability, the agents would need to overcome a credit assignment problem, since it may be difficult to ascertain which agents were responsible for creating good reward signals. Reward shaping may help, but if the world is not fully observable, then again we are left in the dark. If we cannot even observe what the other agents are doing, how can we begin to reason about their role in obtaining the current reward? Our solution relies on its simplicity.

Consider an agent in an MDP, learning to maximize a reward that is a function of its observable state and/or actions. There are many well-studied learning techniques to do this [Sutton and Barto, 1999]. The effects of non-stationarity, partial observability, and global rewards can be thought of as replacing this true reward signal with an alternate signal that is a non-stationary function of the original reward. Think of the difference between learning with a personal coach and learning in a large class where feedback is given only on collective performance. This causes problems for an agent that is trying to use the reward signal to learn an optimal policy for this environment. Since each single agent is unable to observe the entire global state, it might become confused as to whether the changes in its observed reward are due to its own actions or to other external factors, such as the performance of the other agents in the system. A high collective class score may simply reflect that one has smart classmates, rather than whether or not one has learned anything useful. Ideally the agent can recover the original personal reward signal and learn using that signal rather than the global reward signal.

4.2 A simple solution: Filtering the reward signal

These large, partially observable stochastic games, even restricted to fully cooperative games, are clearly a difficult domain in which to learn to behave optimally. However, our approach is to simply abstract away much of the domain's complexity, leaving only enough in our model that we can still achieve reasonable levels of performance. While a more complicated model may be able to perform better, the reasoning is that if we are limited in our computational or observational abilities, then we may simply have to live with some simple, good approximations. Furthermore, it is somewhat surprising that a large number of real-world situations are well-approximated by the simple model we propose in this section.

We show that in many naturally arising situations where an agent attempts to learn a good policy by observing a global, shared reward signal, an effective approach is for this individual agent to model the observed global reward signal as the sum of its own contribution (which is the personal reward signal on which it should base its learning) and a random Markov process (which is the amount of the observed reward due to other agents or external factors). With such a simple model, we can estimate both of these quantities efficiently using an online Kalman filtering process. Many external sources of reward (which could be regarded as noise) can be modeled as or approximated by a random Markov process, so this technique promises broad applicability. This approach is more robust than trying to learn directly from the global reward, allowing agents to learn and converge faster to an optimal or near-optimal policy, sometimes even in domains where convergence was once elusive.

The innovative aspect of our approach is to consider the reward signal as merely a signal that is correlated with our true learning signal. We propose a model that captures the relationship between the true reward and the noisy rewards in a wide range of problems. Thus, without assuming much additional domain knowledge, we can use filtering methods to recover the underlying true reward signal from the noisy observed global rewards.

4.3 Mathematical model

The agent assumes that the world possesses one or more unobservable state variables that affect the global reward signal. These unobservable states may include the presence of other agents or changes in the environment. Each agent models the effect of these unobservable state variables on the global reward as an additive noise process b_t that evolves according to $b_{t+1} = b_t + z_t$, where z_t is a zero-mean Gaussian random variable with variance σ_w. The global reward that it observes if it is in state i at time t is $g_t = r(i) + b_t$, where r is a vector containing the ideal training rewards $r(i)$ received by the agent at state i. The standard model that describes such a linear system is:

$$g_t = Cx_t + v_t, \quad v_t \sim N(0, \Sigma_2)$$
$$x_t = Ax_{t-1} + w_t, \quad w_t \sim N(0, \Sigma_1)$$

In our case, we desire estimates of $x_t = [r_t^T \;\; b_t]^T$. We impart our domain knowledge into the model by specifying the estimated variance and covariance of the components of x_t. In our case, we set $\Sigma_2 = 0$ since we assume no observation noise when we experience rewards; $\Sigma_1(j,j) = 0, j \neq |S| + 1$, since the rewards are fixed and do not evolve over time; $\Sigma_1(|S|+1,|S|+1) = \sigma_w$ since the noise term evolves with variance σ_w. The system matrix is $A = I$, since none of the components of x_t are combined to produce x_{t+1}, and the observation matrix is $C = [0 \;\; 0 \ldots 1_i \ldots 0 \;\; 0 \;\; 1]$ where the 1_i occurs in the i^{th} position when our observed state $s = i$. This corresponds to selecting the correct state and adding in the noise term b.

4.3.1 Kalman filters

Kalman filters [Kalman, 1960] are Bayes optimal, minimum mean-squared-error estimators for linear systems with Gaussian noise. The agent applies the following causal Kalman filtering equations at each time step to obtain maximum likelihood estimates for b and the individual rewards $r(i)$ for each state i given all previous observations. First, the estimate $\hat{x}$ and its covariance matrix P are updated in time based on the linear system model:

$$\hat{x}'_t = A\hat{x}_{t-1} \tag{4.1}$$

$$P'_t = AP_{t-1}A^T + \Sigma_1 \tag{4.2}$$

Then these a priori estimates are updated using the current time period's observation g_t:

$$K_t = P'_tC^T(CP'_tC^T + \Sigma_2)^{-1} \tag{4.3}$$

$$\hat{x}_t = \hat{x}'_t + K_t(g_t - C\hat{x}'_t) \tag{4.4}$$

$$P_t = (I - K_tC)P'_t \tag{4.5}$$

As shown, the Kalman filter also gives us the estimation error covariance P_t, from which we know the variance of the estimates for r and b. We can also compute the likelihood of observing g_t given the model and all the previous observations. This will be handy for evaluating the fit of our model, if needed. We could also create more complicated models if our domain knowledge shows that a different model would be more suitable. For example, if we wanted to capture the effect of an upward bias in the evolution of the noise process (perhaps to model the fact that all the agents are learning and achieving higher rewards), we could add another variable u, initialized such that $u_0 > 0$, modifying x to be $x = [r^T \;\; b \;\; u]^T$, and changing our noise term update equation to $b_{t+1} = b_t + u_t + w_t$. In other cases, we might wish to use non-linear models that would require more sophisticated techniques such as extended Kalman filters.

4.3.2 Q-learning

For the learning mechanism, one possibility is to use a simple tabular Q-learning algorithm [Sutton and Barto, 1999], since we wish to focus our attention on the reward signal problem. Q-learning keeps a "Q-value" for each state-action pair, and proceeds using the following update rule:

$$Q_t(s,a) = (1-\alpha)Q_{t-1}(s,a) + \alpha(r + \gamma \min_{a'} Q_t(s',a')) \quad , \tag{4.6}$$

where $0 < \alpha < 1$ is parameter that controls the learning rate, r is the reward signal used for learning at time t given s and a, $0 < \gamma \leq 1$ is the discount factor, and s, a, and s' are the current state, action, and next state of the agent, respectively. Under fairly general conditions, in a stationary MDP, Q-learning converges to the optimal policy, expressed as

$$\pi(s) = \text{argmax}_a \, Q(s,a) \quad .$$

4.3.3 Model solving

One of the disadvantages of Q-learning is the fact that it must use many iterations to propagate adjustments to the Q-values throughout the state space. During each time period, we only make one update, based on our next state's observations. In order to speed up this value propagation, we can use model-based methods instead. Q-learning and other TD(λ) methods are known as model-free methods, since they do not explicitly try to estimate the transition probabilities between different states given different actions. Instead, they only keep estimates of the rewards for various state, action pairs, and rely on the simulation experience to propagate these values in the correct probabilistic way, since our simulation experience will be generated from the underlying transition probabilities.

If we estimate these transition probabilities, we can make the process more efficient by obviating the need for many iterations of simulations to propagate these values. Instead, once we have estimated the transition probabilities accurately, we can simply use value iteration or policy iteration to produce the optimal policy given our estimated rewards and transition probabilities. If we are using actual robots in the real world, this is clearly the route we want to take, since acquiring experience in the world is very costly. In simulation, there is less need to use model solving, since simulation experience is in comparison quite cheap. However, we will see that using a good model-based technique will still allow us to find an optimal policy in fewer iterations. The number of computations made during each iterations is larger using the model-based technique, however, since one call of a procedure such as value iteration requires many computation cycles.

For our experiments, we use a simplified version of Kearns and Singh's E3 algorithm [Kearns and Singh, 1998]. Like other model-based learning algorithms, it attempts to estimate the transition probability matrix of the domain. Its differentiating characteristic is that it can be shown to produce an ϵ-optimal policy in polynomial time. However, it does require several expensive computations of value iteration during many of its iterations. Thus, in our simplified version, rather than actually using a very long mixing time λ that is computed from the domain model, we simply choose an arbitrary λ that is fairly large. Furthermore, instead of explicitly solving for the optimal exploration path, we will usually simply choose the action that has been least explored in the past. For the domains we investigate, this simplified version of E3 runs much faster and seems to perform adequately well.

4.4 Algorithm

Like any good student, the filtering learning agent chooses to accept well-deserved praise from its teacher and ignore over-effusive rewards. The good student does not update his behavior at every time step, but only upon observing relevant rewards. Getting an A in a class with an easy professor should not convince me that I have good study habits! The question remains: How does an agent decide upon the relevance of the rewards it sees? We have proposed a model in which undeserved rewards over time are captured by a Markov random process b. Using observations from previous states and actions, an agent can approach this question from two perspectives. In the first, each time the agent visits a particular state s, it should gain a better sense of the evolution of the random variable b between its last visit and its current visit. Secondly, given an estimate of b_t during a visit to s at time t, it has a better idea of the value of b_{t+1} when it visits s' at time $t+1$. These are the ideas captured by the causal Kalman filter, which only uses the history of past states and observations to provides estimates of $r(i)$ and b.

The agent follows this simple algorithm:

1. From initial state s_0, take some action a, transition to state i, and receive reward signal g_0. Initialize $\hat{x}_0(i_0) = g_0$ and $\hat{x}_0(|S|+1) = b_0 = 0$, since $b_0 = 0$.

2. Perform a Kalman update using equations 4.1-4.5 to compute the current vector of estimates $\hat{x}$, which includes a component that is the reward estimate $\hat{r}(s_0)$, which will simply equal g this time.

3. From the current state i at time t, take another action with some mix of exploration and exploitation; transition to state j, receiving reward signal g_t. If this is the first visit to state i, initialize $\hat{x}_t(i) = g_t - \hat{b}_{t-1}$.

4. Perform a Kalman update using equations 4.1-4.5 to compute the current vector of estimates $\hat{x}$, which includes a component that is the reward estimate $\hat{r}(i)$.

5. Update the Q-table using $\hat{r}(i)$ in place of r in equation 4.6; return to Step 3.

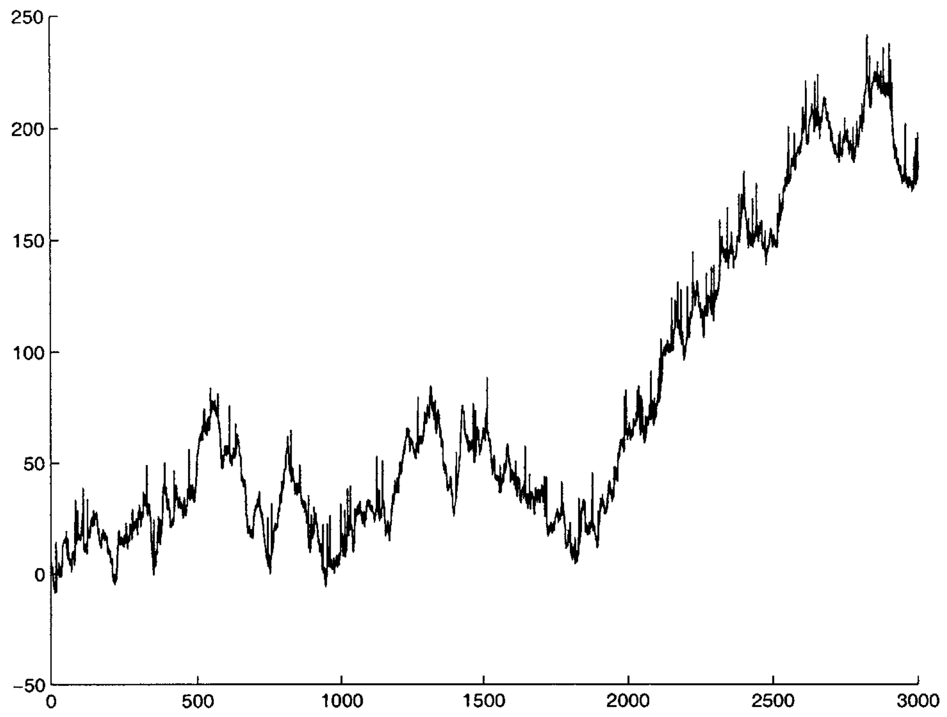


Figure 4-1: As the agent in the idealized single-agent grid world is attempting to learn, the reward signal value (y-axis) changes dramatically over time (x-axis) due to the noise term. While the true range of rewards in this grid world domain only falls between 0 and 20, the noisy reward signal ranges from -10 to 250, as shown in the graph at left.

The advantage of the Kalman filter is that it requires a constant amount of memory – at no time does it need a full history of states and observations. Instead, it computes a sufficient statistic during each update, x and P, which consists of the maximum likelihood estimate of r and b, and the covariance matrix of this estimate. Thus, we can run this algorithm online as we learn, and its speed does not deteriorate over time.

4.5 Experiments

If the world dynamics match the linear model we provide the Kalman filter, then certainly this method will work well. The interesting question concerns situations in which the actual dynamics are clearly different from the model, and whether this

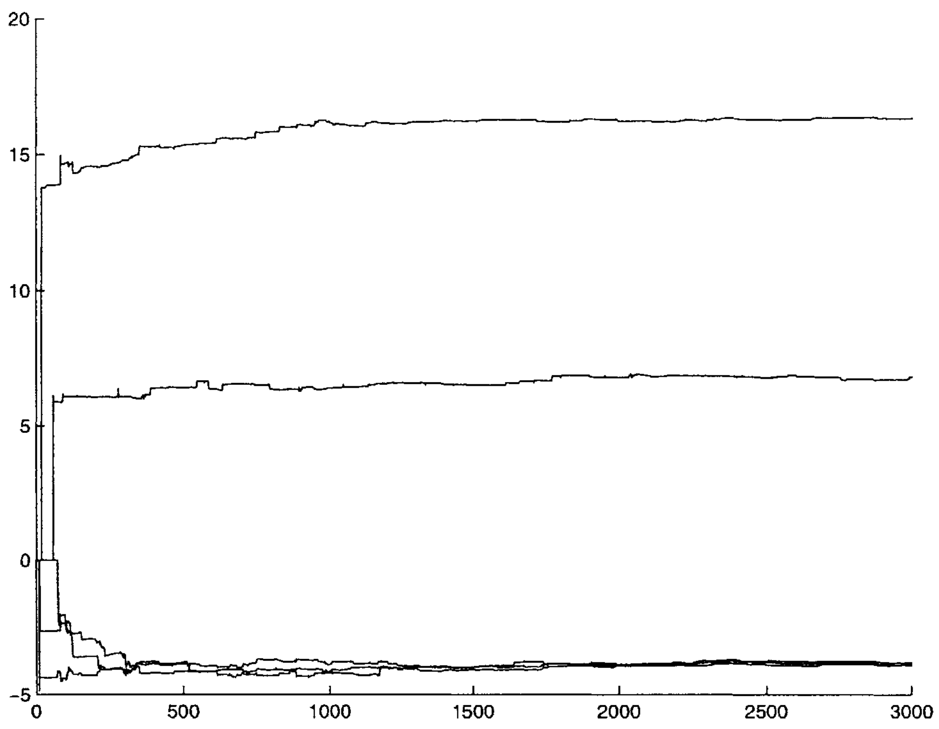


Figure 4-2: Given the noisy signal shown in Figure 4-1, the filtering agent is still able to learn the true underlying rewards, converging to the correct relative values over time, as shown in the middle graph.

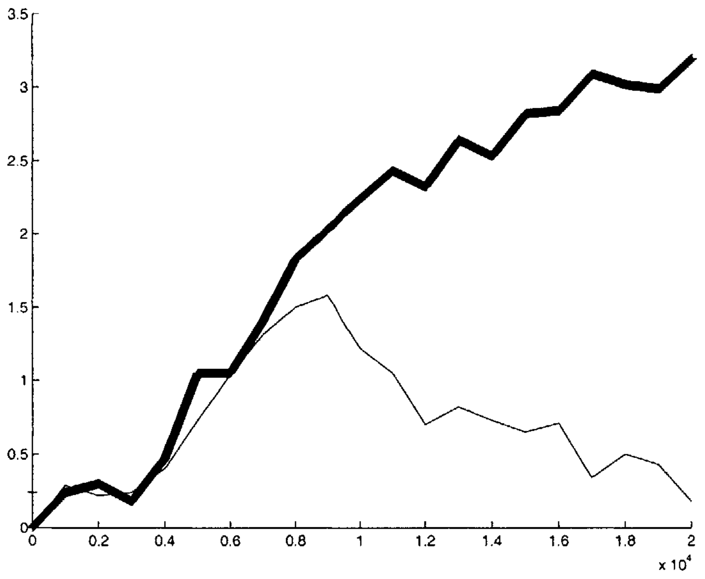


Figure 4-3: The filtering learning agent (bold line) accrues higher rewards over time than the ordinary Q-learner (thin line), since it is able to converge to an optimal policy whereas the non-filtering Q-learner remains confused.

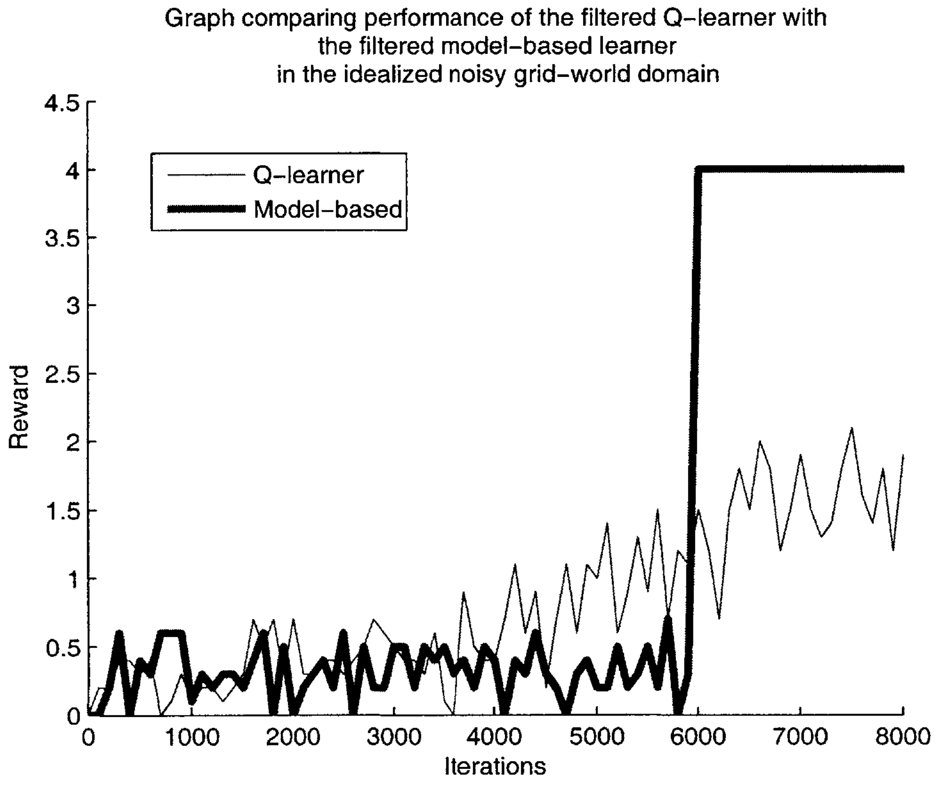


Figure 4-4: The filtering model-based learning agent (bold line) is able to learn an optimal policy much quicker than the filtering Q-learning agent (thin line), since it is able to use value iteration to solve for the optimal policy once the personal reward signals have been accurately estimated.

filtering agent will still learn good, or even optimal, policies in such cases. This section examines the efficacy of the filtering learning agent in several different domains: (1) a single agent domain in which the linear system describes the world perfectly, (2) a single agent domain where the noise is manually adjusted without following the model, (3) a multi-agent setting in which the noise term is meant to encapsulate presence of other agents in the environment, and (4) a more complicated multi-agent setting that provides an abstraction of a mobile ad-hoc networking domain in which mobile agent nodes are trying to maximize total network performance.

For ease of exposition, all the domains we use in this chapter are variants of the basic grid-world domain shown in Figure 4-5 and described in various reinforcement learning texts such as [Sutton and Barto, 1999]. The agent is able to move North, South, East, or West from its present position, and most transitions give the agent zero reward, except all actions from state 6 move the agent directly to state 10 with a reward of 20, and all actions from state 16 move the agent directly to state 18 with a reward of 10. Bumps into the wall cost the agent -1 in reward and move the agent nowhere. We use a discount factor of 0.9.

4.5.1 Idealized noisy grid world

To demonstrate the basic feasibility of our filtering method, we first create a domain that follows the linear model of the world given in Section 4.3 perfectly. That is, in each time step, a single agent receives its true reward plus some noise term that evolves as a Markov random process. To achieve this, we simply add a noise term to the grid world domain given in Figure 4-5. As shown in Figure 4-1, an agent acting in this domain will receive a large range of reward values due to the evolving noise term. In the example given, sometimes this value ranges as high as 250 even though the maximum reward in the grid world is 20 – the noise term contributes 230 to the reward signal! A standard Q-learning agent does not stand a chance at learning anything useful using this reward signal. However, the filtering agent can recover the true reward signal from this noisy signal and use that to learn. Figure 4-3 shows that the filtering agent can learn the underlying reward signals, converging to these values relatively quickly. The graph to the right compares the performance of the filtering learner to the normal Q-learner, showing a clear performance advantage.

Furthermore, we can compare the performance of the filtering Q-learning agent with the performance of the filtering simplified E3 learning agent. As shown in Figure 4-4, the filtering model-based learning agent is able to learn an optimal policy much quicker than the filtering Q-learning agent, since it is able to use value iteration to solve for the optimal policy once the personal reward signals had been accurately estimated. From Figure 4-2, we know that the personal reward signal becomes stable after only a small number of iterations. The model-based algorithm is able to solve for the optimal policy as soon as this happens, thus learning the optimal policy much quicker than the model-free Q-learner.

The observant reader may note that the learned rewards do not match the true rewards specified by the grid world. Specifically, they are offset by about -4. Instead of mostly 0 rewards at each state, the agent has concluded that most states produce reward of -4. Correspondingly, state 6 now produces a reward of about 16 instead of 20. Since Q-learning will still learn the correct optimal policy subject to scaling or translation of the rewards, this is not a problem. This oddity is due to the fact that our model has a degree of freedom in the noise term b. Depending on the initial guesses of our algorithm, the estimates for the rewards may be biased. If most of initial guesses for the rewards underestimated the true reward, then the learned value will be correspondingly lower than the actual true value. In fact, all the learned values will be correspondingly lower by the same amount.

To further test our filtering technique, we next evaluate its performance in a domain that does not conform to our noise model perfectly, but which is still a single agent system. Instead of an external reward term that evolves according to a Gaussian noise process, we adjust the noise manually, introducing positive and negative swings in the reward signal values at arbitrary times. The results are similar to those in the perfectly modeled domain, showing that the filtering method is fairly robust.

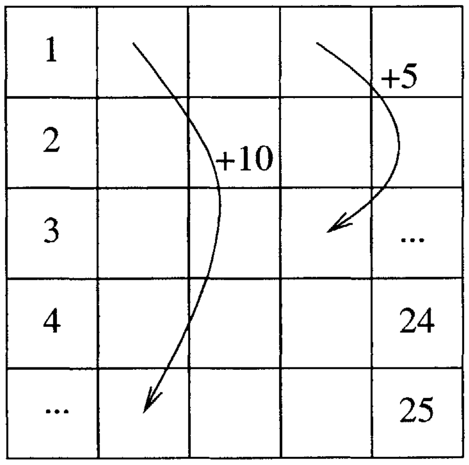


Figure 4-5: This shows the dynamics of our 5x5 grid world domain. The states correspond to the grid locations, numbered 1,2,3,4,...,24,25. Actions move the agent N,S,E, or W, except in states 6 and 16, where any action takes the agent to state 10 and 18, respectively, shown by the curved arrows in the figure at left.

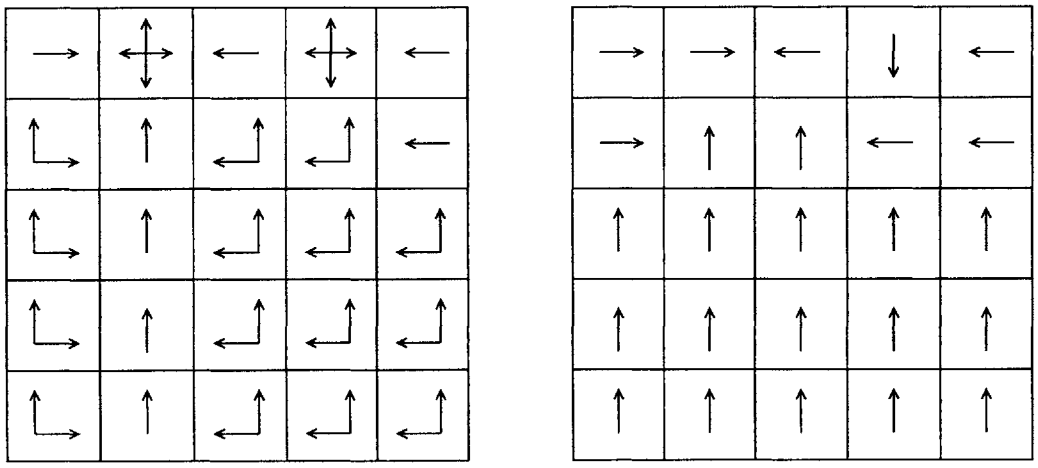


Figure 4-6: The optimal policy for the grid world domain is shown at left, where multiple arrows at one state denotes indifference between the possibilities. A policy learned by our filtering agent is shown at right. The learning algorithm does not explicitly represent indifference, and thus always picks one action to be the optimal one.

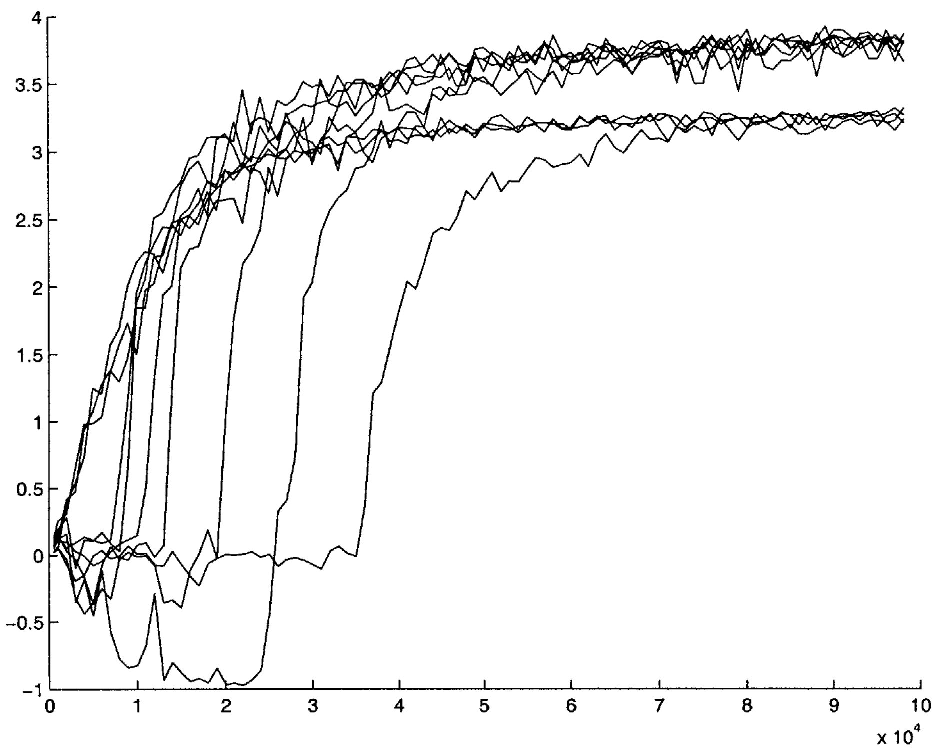


Figure 4-7: Filtering agents are able to distinguish their personal rewards from the global reward noise, and thus able to learn optimal policies and maximize their average reward over time in a ten-agent grid-world domain.

4.5.2 Multi-agent grid world

The most interesting case occurs when the domain noise is actually caused by other agents learning in the environment. This noise will not evolve according to a Gaussian process, but since the filtering method is fairly robust, we might still expect it to work. If there are enough other agents in the world, then the noise they collectively generate may actually tend towards Gaussian noise. Here we focus on smaller cases where there are 6 or 10 agents operating in the environment. We modify the grid world domain to include multiple simultaneously-acting agents, whose actions do not interfere with each other, but whose reward signal now consists of the sum of all the agents' personal rewards, as given in the basic single agent grid world of Figure 4-5.

We again compare the performance of the filtering learner to the ordinary Q-learning algorithm. As shown in Figure 4-9, most of the filtering learners quickly

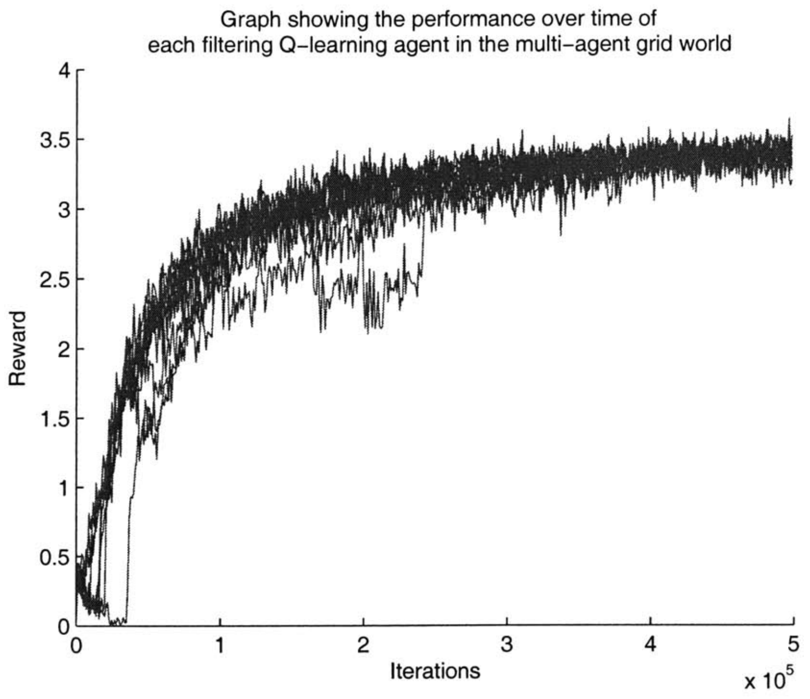


Figure 4-8: Another graph showing that filtering Q-learners converge. Unlike Figure 4-7, the exploration rate decay is slowed down, thus allowing all the agents to converge to the optimal policy, rather than being stuck in the sub-optimal policy.

converge to the optimal policy. Three of the 10 agents converge to a suboptimal policy that produces slightly lower average rewards. However, this artifact is largely due to our choice of exploration rate, rather than a large error in the estimated reward values. The standard Q-learning algorithm also produces decent results at first. Approximately half of the agents find the optimal policy, while the other half are still exploring and learning. An interesting phenomenon occurs when these other agents finally find the optimal policy and begin receiving higher rewards. Suddenly the performance drops drastically for the agents who had found the optimal policy first. Though seemingly strange, this provides a perfect example of the behavior that motivates our method. When the other agents learn an optimal policy, they begin affecting the global reward, contributing some positive amount rather than a consistent zero. This changes the world dynamics for the agents who had already learned the optimal policy and causes them to "unlearn" their good behavior.

The unstable dynamics of the Q-learners could be solved if the agents had full

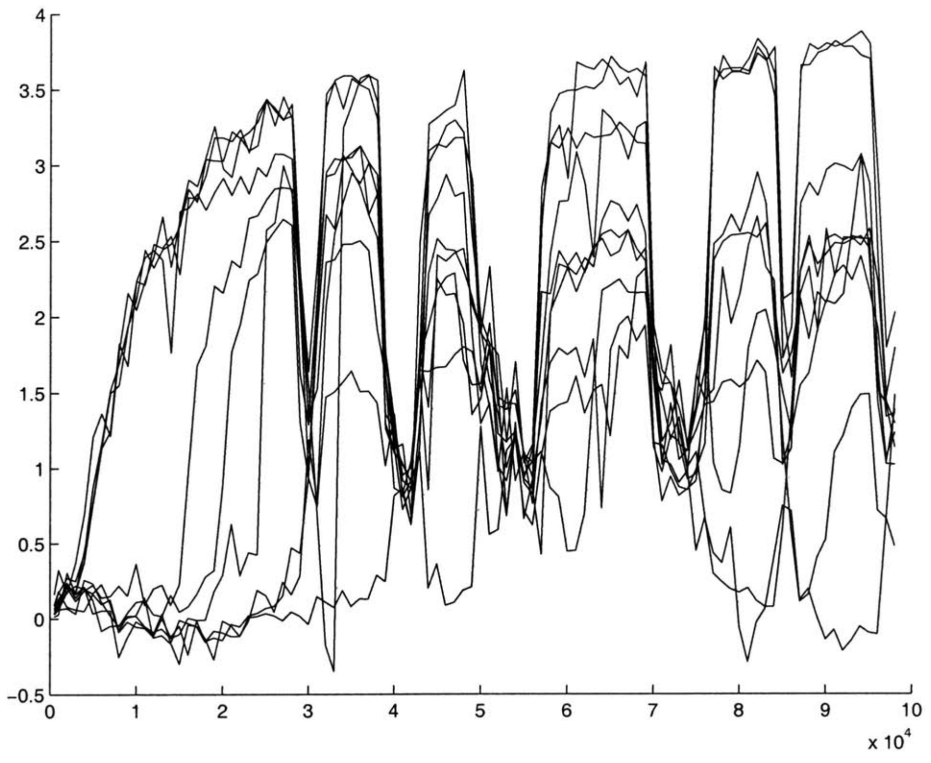


Figure 4-9: In contrast to Figure 4-7, ordinary Q-learning agents do not process the global reward signal and can become confused as the environment changes around them. Graphs show average rewards (y-axis) within 1000-period windows for each of the 10 agents in a typical run of 10000 time periods (x-axis).

observability, and we could learn using the joint actions of all the agents, as in the work of Claus and Boutilier [1998]. However, since our premise is that agents have only a limited view of the world, the Q-learning agents will only exhibit convergence to the optimal policy if they converge to the optimal policy simultaneously. This may take a prohibitively long time, especially as the number of agents grows.

It is instructive to note that this type of oscillation does not occur if we use unfiltered model-based techniques. Over a long enough period of time, the model-based technique builds up a good estimate of the average reward signal at each state, which it then uses as the personal training signal to solve for the optimal policy. However, as shown in Figure 4-10, the filtered version of the model-based learner is still able to find the optimal policy more quickly, since the filter is able to recover the personal training signal more quickly than simple averaging. This is discussed further

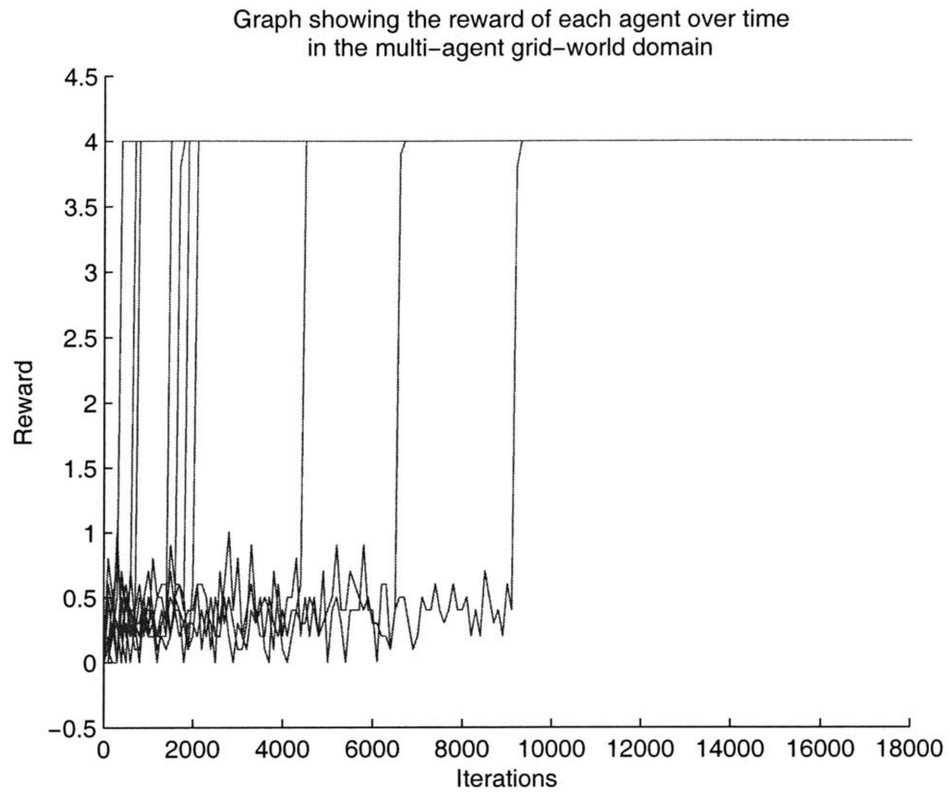


Figure 4-10: Once again, the filtered model-based agent is able to learn an optimal policy in fewer simulation iterations than the filtered model-free Q-learning agent. Note that the scale of the x-axis in this figure is different from the previous figures.

in Section 4.6.3.

Once again, as in the previous experimental domain, the model-based method finds the optimal policy in a much smaller number of iterations than the Q-learner. Note that the scale of the x-axis in Figure 4-10 is much larger than in Figure 4-7.

4.5.3 Mobilized ad-hoc networking

Finally, we apply our filtering method to a more realistic domain. We investigate the mobilized ad-hoc networking domain by simplifying and adapting it to the grid world. As discussed earlier, mobilized ad-hoc networking provides an interesting real-world environment that illustrates the importance of reward filtering due to its high degree of partial observability and a reward signal that depends on the global state.

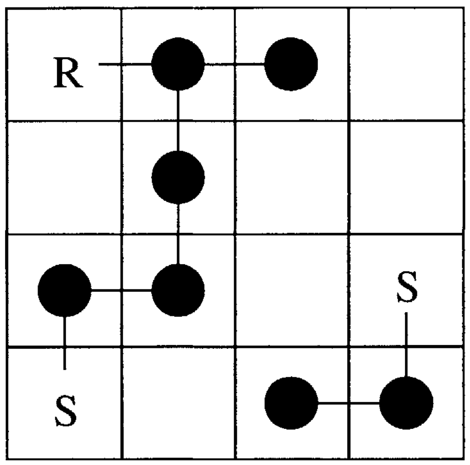


Figure 4-11: A snapshot of the 4x4 adhoc-networking domain. S denotes the sources, R is the receiver, and the dots are the learning agents, which act as relay nodes. Lines denote current connections. Note that nodes may overlap.

In this domain, there are a number of mobile nodes whose task is to move in such a way as to optimize the connectivity (performance) of the network. Chang et al. [2004b] cast this as a reinforcement learning problem. As the nodes move around, connections form between nodes that are within range of one another. These connections allow packets to be transmitted between various sources and receivers scattered among the nodes. The nodes are limited to having only local knowledge of their immediate neighboring grid locations (rather than the numbered state locations as in the original grid world), and thus do not know their absolute location on the grid. They are trained using a global reward signal that is a measure of total network performance, and their actions are limited functions that map their local state to N, S, E, W movements. We also limit their transmission range to a distance of one grid block. For simplicity, the single receiver is stationary and always occupies the grid location (1,1). Source nodes move around randomly, and in our example here, there are two sources and eight mobile agent nodes in a 4x4 grid.

This setup is shown in Figure 4-11, and the graph shows a comparison of an ordinary Q-learner and the filtering learner, plotting the increase in global rewards over time as the agents learn to perform their task as intermediate network nodes. The graph plots average performance over 10 runs, showing the benefit of the filtering

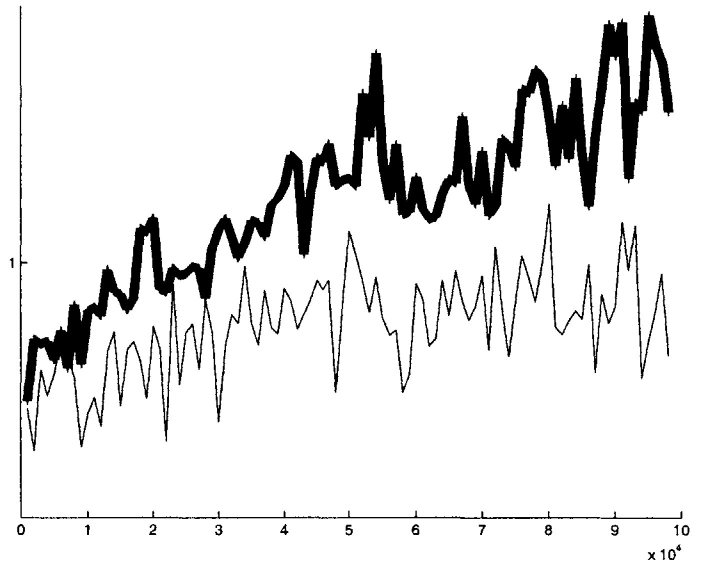


Figure 4-12: Graph shows average rewards (y-axis) in 1000-period windows as filtering Q-learner (bold line) and unfiltered Q-learner (thin line) agents try to learn good policies for acting as network nodes in the ad-hoc networking domain. The filtering agent is able to learn a better policy, resulting in higher network performance (global reward). Graph shows the average for each type of agent over 10 trial runs of 100000 time periods (x-axis) each.

process. Again, the model-based learner is able to learn a better policy within fewer simulation iterations. Similar experiments for larger grid sizes (5x5 and 6x6) with larger numbers of sources and mobile agent nodes also produced similar results.

4.6 Discussion

Clearly, this filtering approach will not produce good results for any given multi-agent domain. The experiments in the previous section show that the filtering approach, despite its restrictive assumptions about the nature of the global reward signal, actually performs well in a variety of domains where we would expect that these assumptions are violated. In this section, we will explore the reasons why this approach worked well in these domains, and discuss other domains where this approach does not work.

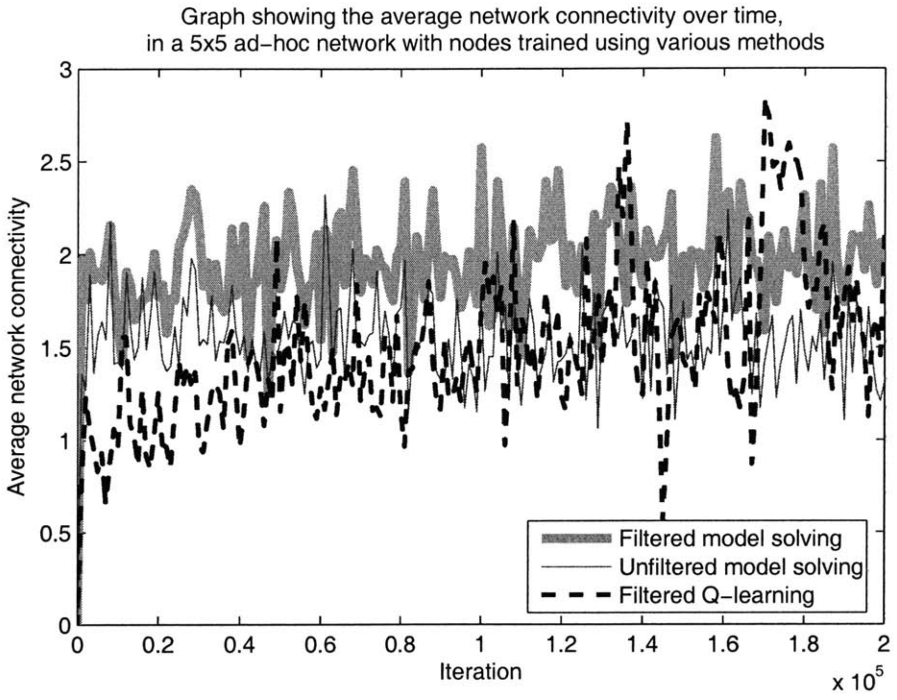


Figure 4-13: Graph shows average rewards (y-axis) in 1000-period windows as filtering model-based learners (bold line), filtered Q-learners (dashed line), and unfiltered model-based learners (thin line) try to learn good policies for acting as network nodes in the ad-hoc networking domain. The trials are run on a 5x5 grid, with 4 sources, 1 receiver, and 13 learning nodes.

4.6.1 Limitations

First of all, this filtering approach was designed to deal with cooperative multiagent domains where a single global reward signal is given to all the agents. We could, of course, try to apply this technique to noncooperative situations where the multiple agents in the environment do not necessarily receive the same reward at each time period. The reward filtering would still try to achieve the same goals, i.e. it would attempt to filter out the effect of the opponents and of the unobserved portion of the environment on the unobserved reward signal. However, this does not work since the opponents may be actively adapting to our action choices. Thus, and action that is a good action in the current time period may no longer be a good action in some following time period. The filtering method, as presented in this chapter, only allows us to learn a static policy that maps states to actions. The reward distribution for

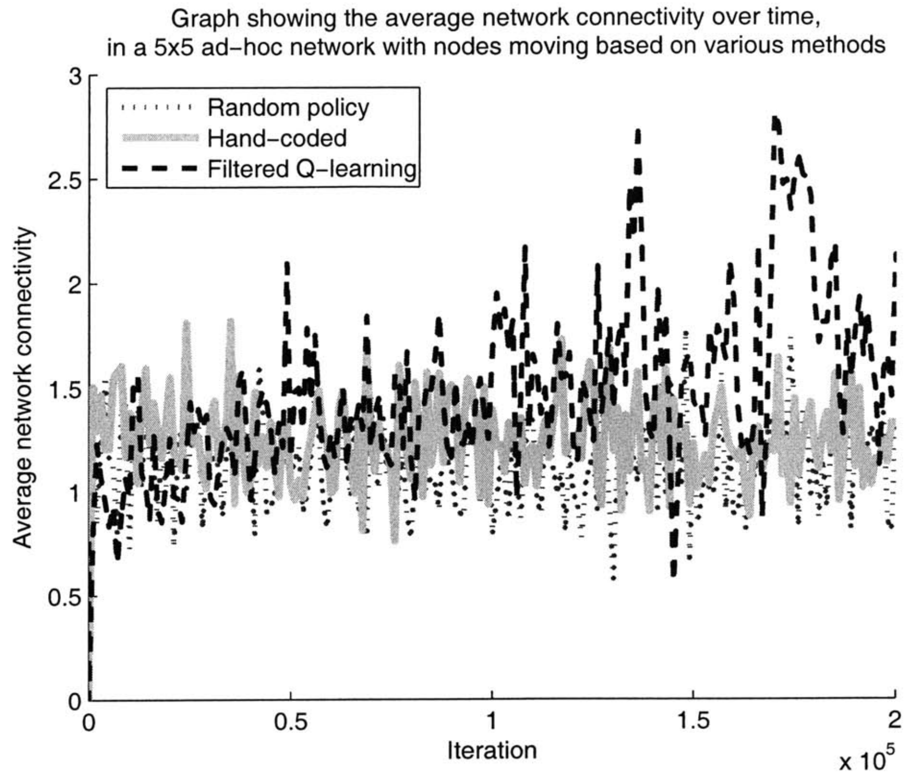


Figure 4-14: Graph shows average network connectivity in 1000-period windows as filtered Q-learning agents (dashed line), hand-coded agents (bold line), and random agents (dotted line) act as network nodes in the ad-hoc networking domain. The trials are run on a 5x5 grid, with 4 sources, 1 receiver, and 13 learning nodes.

each observed state, action pair needs to remain constant or at least change only very slowly.

While this may be true for cooperative domains, it is highly unlikely to be the case in competitive situations. The filtering method would thus be just as confused as any other $\mathcal{H}_\infty \times \mathcal{B}_0$ player, if playing against a more intelligent opponent that its assumptions allow. Just as described in Chapter 2, however, we could expand the state space to include observation histories. In a two-player game, filtering would be longer make sense. Filtering attempts to remove the complexity of the environment; here the environment is already as simple as it can be. Filtering also attempts to remove the complexity of the other agents' action choices; here these choices are tightly correlated with our own choices, and thus cannot be removed using our described filtering method without major modifications. We will discuss better ways of dealing with this type of situation in the next chapter.

Another situation where the filtering approach will likely fail is in a multi-agent domain where they are a very large number of agents. With a large number of other agents in the environment, it is more difficult for a single agent to estimate the effect of its actions on the global reward signal, since any incremental change in the reward signal would likely get washed out by all the noise. Basically, the signal to noise ratio will drop as the number of agents grows. Thus, it won't work for huge numbers of agents, even though the "noise" may begin to more closely resemble Gaussian noise due to the Central Limit Theorem, if we assume that the rewards generated by the different agents are independently drawn.

4.6.2 Model assumptions

On the other hand, there are many situations in which the filtering method will produce good results, even though we do not expect that our restrictive model assumptions hold in these situations. For example, in our experiments in Section 4.5, the filtering method allowed our learning algorithms to learn good policies even though we do not expect the noise model to hold in these environments. However, as we demonstrate in this section, it may be surprising that the number of environments in which is noise model does indeed hold is quite large. This is not to say that there aren't also many environments in which filtering is not applicable. For example, in many single-agent POMDPs, such as those in Loch and Singh [1998], the filtering technique cannot remove any "noise" and learn a good reactive policy since the learning depends heavily on the learning and exploration technique used, which affect the estimated performance of any policy by affecting the state occupation frequencies of the underlying MDP. Furthermore, in adversarial environments, the filtering technique will also fail, since the "noise" is actually our adversary's response to our actions and is thus very likely to depend heavily on our action choices and change over time.

The intuition is that as long as the effect of the other agents in the environment on the change in the global reward signal is somewhat independent of our agent's actions,

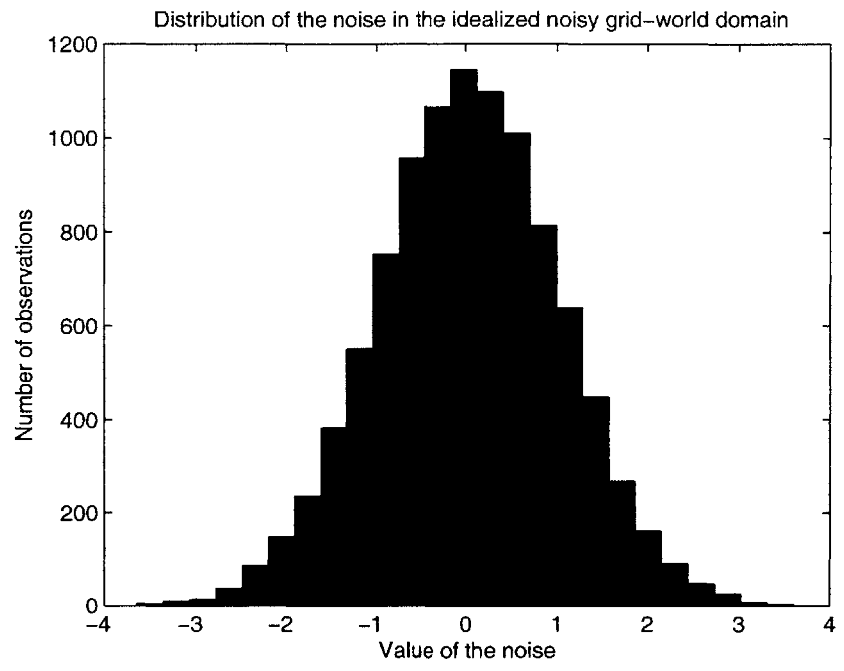


Figure 4-15: This graph shows the distribution of the noise term over time in the idealized noisy grid world simulation. The graph closely mimics that of a standard normal distribution, since the noise is drawn from a normal distribution.

then this effect can be approximated reasonably well as a Gaussian distribution. The effect of the other agents in each time period is simply a random draw from the Gaussian distribution in our model.

In the idealized noisy grid world domain, the noise is exactly represented by a Gaussian distribution. Figure 4-15 shows that over time, the random draws from this distribution indeed produce a histogram that closely resembles the normal distribution. This histogram counts the number of observations in each range of the noise values over time. Each time period produces one observation.

While we might have expected Figure 4-15 to resemble a Gaussian distribution since the idealized noisy grid world uses such a distribution to produce the noise in that domain, it comes as more of a surprise that a similar histogram produced for the multiagent grid world also resembles a Gaussian. Figure 4-16 shows such a histogram. Recall that the noise in this domain, from the perspective of agent i, corresponds to the change over time of the sum of the other agents' rewards, i.e.

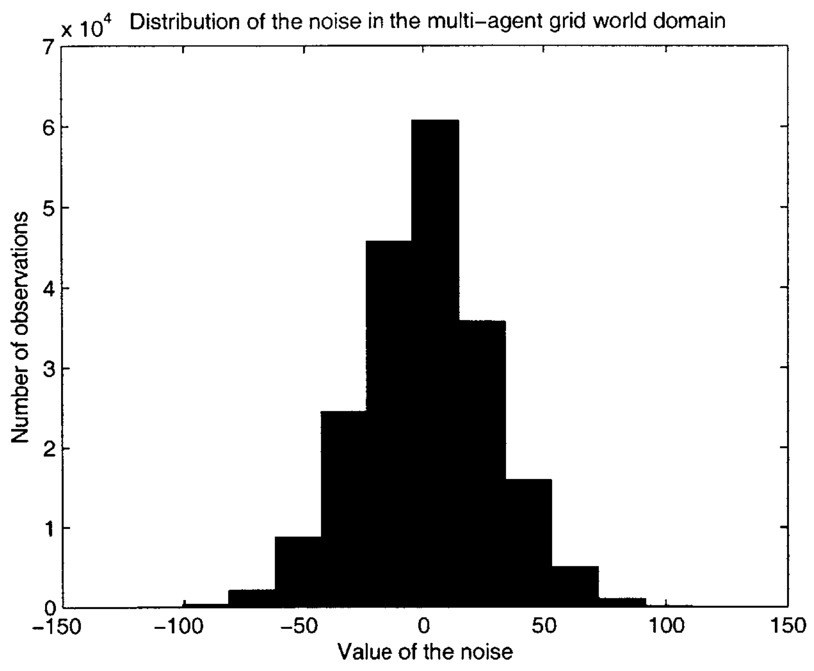


Figure 4-16: This graph shows the distribution of the noise term over time in the multi-agent grid world simulation. As we can see, the distribution still resembles a standard normal distribution even though the noise is no longer drawn from such a distribution but is instead generated by the other agents acting in the domain.

$b_t = \sum_{j \neq i} r_t^j - \sum_{j \neq i} r_{t-1}^j$. We can see that the histogram of this noise resembles a Gaussian distribution. To be more precise, we also measure the mean, variance, skew, and kurtosis of the noise distribution produced in this multi-agent grid world domain. In order for this distribution to correspond to our model, it should have zero mean, positive variance, and zero skew and kurtosis. Table 4.1 shows these statistics. We note that up to the third moment, the noise distribution in the multi-agent grid world corresponds well to the Gaussian.

Finally, we consider the noise distribution in the mobilized ad-hoc network scenario. In this domain,the value of the noise term is less clearly defined, since the global reward signal is no longer a simple sum of one agent's personal reward and some other term. Thus, in order to measure the noise in each time period, we will need to estimate it based on our global knowledge of the network state. Using this global knowledge, we need to construct an estimate of what the single agent's personal

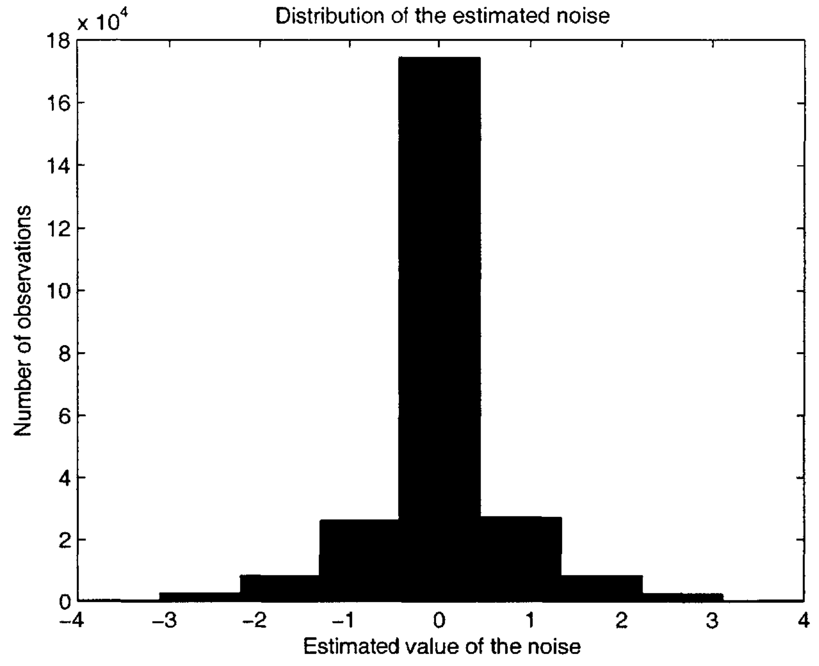


Figure 4-17: This graph shows the distribution of the estimated noise term over time in the multi-agent ad-hoc networking simulation. This distribution only vaguely resembles a normal distribution, but is still symmetric about its mean and thus has low skew.

reward ought to be.

For a particular mobile agent node, we assign a score of +1 for each source for which it serves as a node in the shortest path between that source and a receiver. Its personal reward signal is then approximated as a sum of these scores. This signal attempts to reward mobile agent nodes only when they are actively participating in helping to make a connected network. Clearly this is not necessarily the best estimate to use, since we might imagine occasions when a non-participating node is actually playing an important role by seeking out other unconnected sources, or waiting around in a location where a disconnect is likely to occur soon. However, for the purposes of gaining a clearer picture of the fit for our noise model to the actual environment, our coarse approximation suffices.

Figure 4-17 shows the distribution of the estimated noise term, which is calculated by subtracting the approximated, heuristic personal reward signal from the observed

	Mean	Variance	Skew	Kurtosis
Model assumptions	0	any	0	0
Idealized noisy grid world	0.016	1.017	-0.0166	3.031
Multi-agent grid world	0.000	817.400	0.0059	3.259
Mobilized ad-hoc networking	0.000	0.720	-0.0664	7.501

Table 4.1: Table showing the mean, variance, skew, and kurtosis of the noise distribution measured in the various experimental domains.

global reward signal. Like a Gaussian distribution, it is symmetric, but its shape is slightly different from a Gaussian, as reflected by its positive kurtosis. Since its mean is zero, its kurtosis should also be zero if it were a standard Gaussian distribution. These statistics are shown in Table 4.1. Another reason for the larger kurtosis in this domain is that the potential values for the noise term are discretized, and can only take on a small number of values. For s sources, there are only $2s + 1$ possible values for the noise term. We ran experiments with $s = 3, 4, 5$ and produced similar results in each case.

These measurements show that despite an apparent lack of correspondence between some of these domains and our restrictive noise model, these domains actually correspond to our noise model fairly well. Our filtering technique is able to take advantage of this correspondence and learn good policies using the filtered rewards.

4.6.3 Averaging rewards

One final question arises in this situation. Since we know that the noise term has zero mean, we might think that we could simply average the global reward signal over time and produce a usable training signal in this way. This averaging method would improve upon the nave method of simply training on the global reward signal since it is able to estimate a more stable training signal, preventing the learning agent from becoming confused by the fluctuations in the global signal. While this method may work in the very long run, our filtering technique is much more efficient because it is able to take advantage of the correlations of the noise term estimates between different states of the environment. Thus the filtering technique arrives at a good

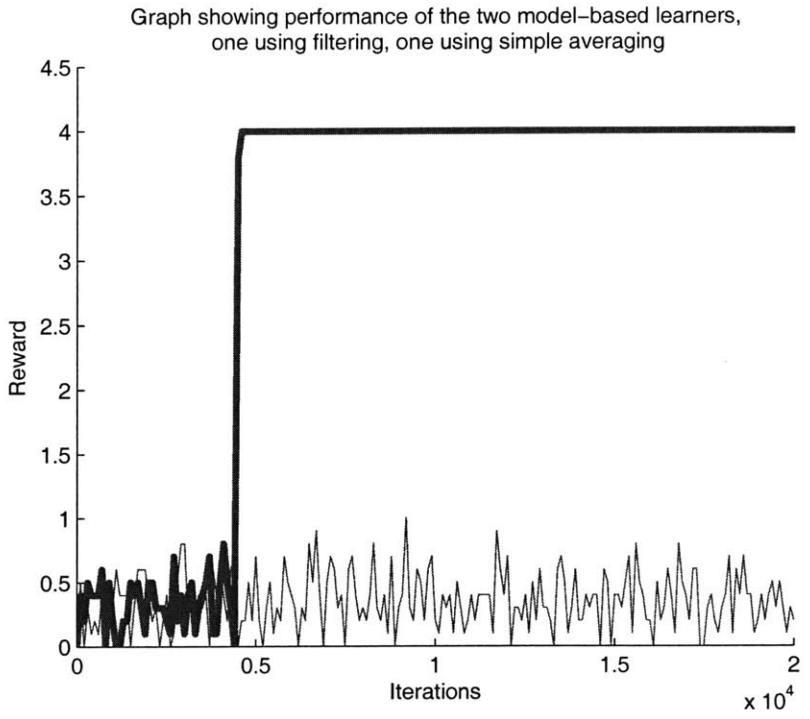


Figure 4-18: This graph shows the performance of a model-based learned using a filtered reward signal (bold line) vs using an averaged reward signal (thin line). The averaging method does not work at all within the 25,000 time periods shown here.

estimate of the personal reward signal much more quickly than a simple averaging method.

Figure 4-18 compares the averaging method with the filtering method in the multi-agent grid-world domain. As we can see, the averaging method does not work at all. The reason is that the zero mean noise may skew the average rewards if the agent has state occupancy frequencies that favor a certain set of states. Thus, in the medium run, the averages will not be centered around the true reward. In the very long run, the expected average would be centered around the true reward, but even so, that would require many many independent trials for this to be true.

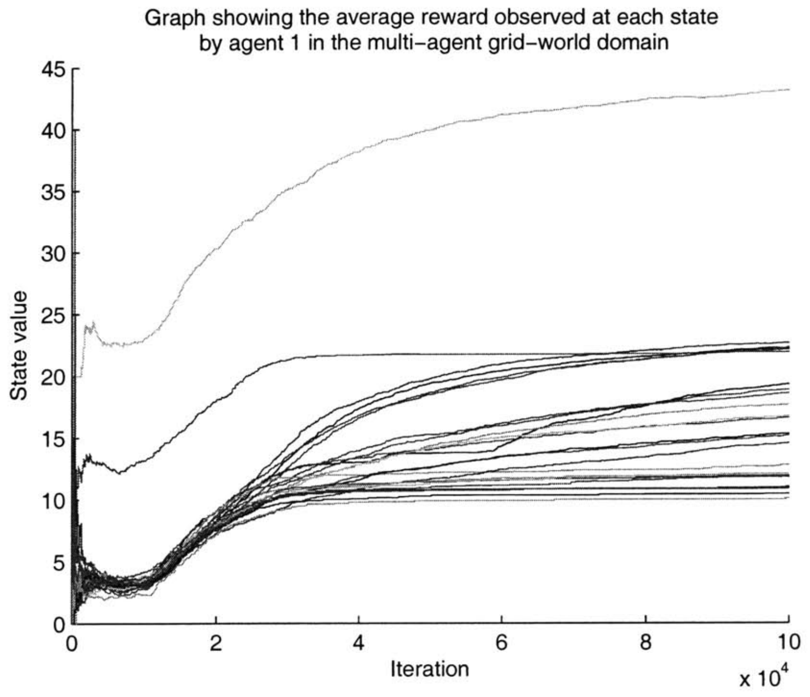


Figure 4-19: Average rewards fluctuate over time, rendering the average reward an inappropriate training signal since it does not give the agents a good measure of their credit as a portion of the global reward signal.

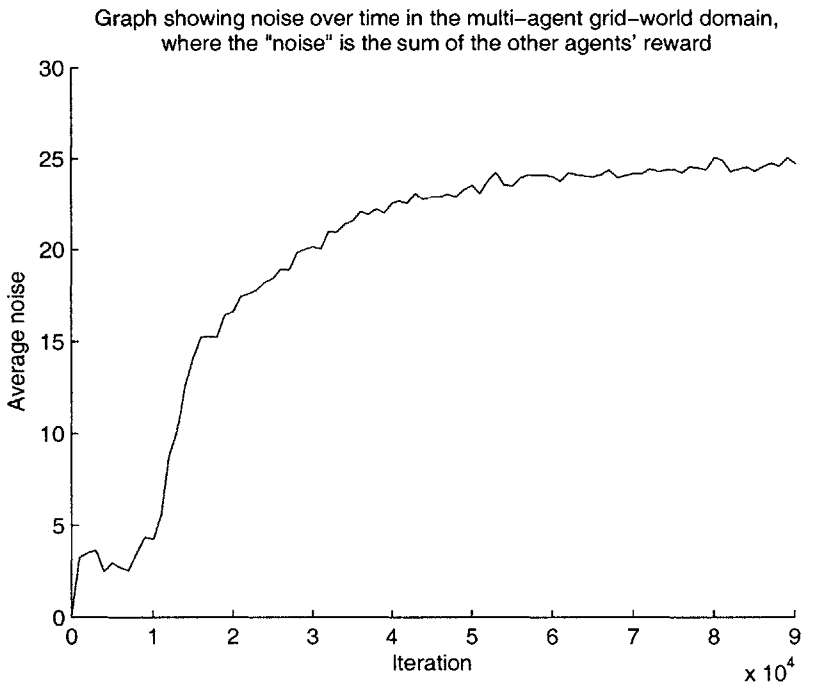


Figure 4-20: This graph shows the noise term evolving over time. While the additive term in each time period has a mean of zero, clearly the noise term b_t increased as the agents learn good policies for behavior.

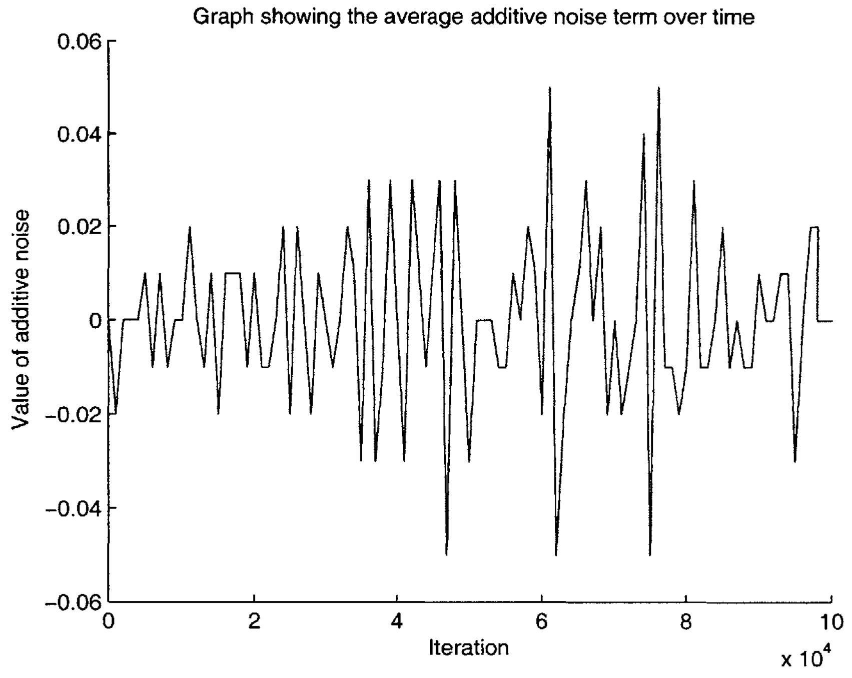


Figure 4-21: This graph shows the average additive noise over time. While the mean is very near zero, small deviations over time produce a noise term b_t that grows away from zero over time.

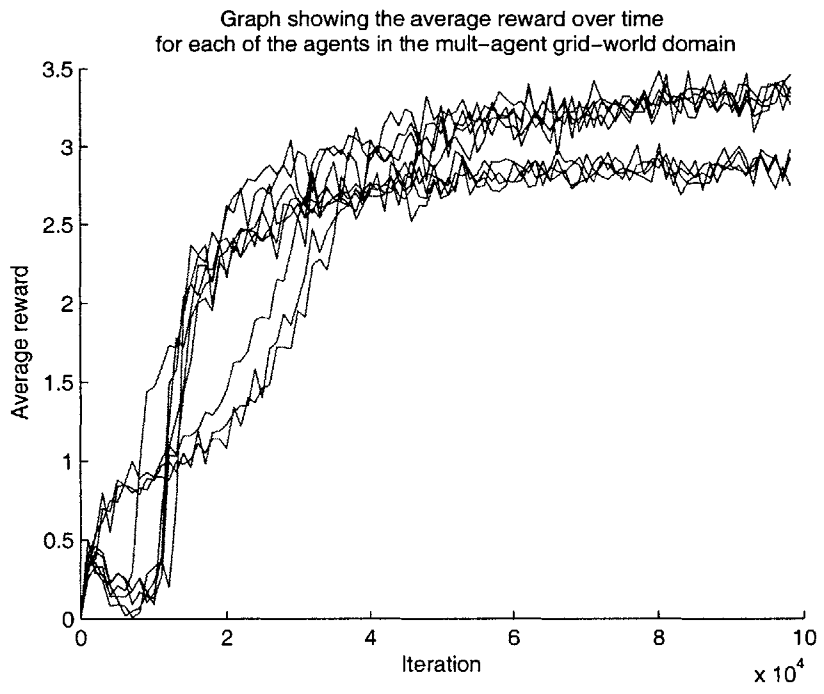


Figure 4-22: The source of the growing noise is clear from this graph. In these experiments, we use filtered learning, and the agents are learning better policies over time, thereby increasing the average global reward signal over time.

Chapter 5

Learning with uncertain models

As we have seen, the environment may become too complex for filtering alone to produce good results in a reasonable amount of time. Furthermore, if the environment is adaptive and adversarial, then filtering alone has no hope of solving our problems. In this chapter, we will introduce the concept of hedged learning, which is able to guarantee a high level of performance against any arbitrary opponent.

Unlike the filtering technique, where we take advantage of a very simple model assumption, using hedged learning we are able to consider different possible assumptions about the world. We might have several different models, all of which has the potential to best explain the world we observe. Ideally, we already know which model best suits the domain at hand, but realistically this is often not the case. Thus, we need an algorithm to behave well in the domain, while learning which model is best, and learning an even better policy for behaving in the world using that best-fit model. We will often refer to the "best model" and the "policy learned from the best model" interchangeably.

Our performance may depend greatly on the appropriateness of the models we have chosen. In partially observable environments, where we have a choice of features and may need to use memory to fully capture the underlying state space, this can be a significant barrier to producing good learning results. Thus, if a particular model does not seem to be producing good behavior, we should ignore its predictions and use a different, better performing model.

Finally, we may be uncertain about the appropriateness of different kinds of learning algorithms for a given domain. Again, since the environment is only partially-observable, simple methods such as Q-learning probably will not do well. However, if there is a good reactive policy that can be learned, it may actually succeed. Algorithms such as policy gradient ascent may also have their shortcomings in these domains, since they are only guaranteed to find local optima. Again, sometimes they may work quite well, especially if we can initialize them to begin searching in an appropriate part of the policy space. Finally, we may need more sophisticated methods such as using a finite-state controller paired with a policy gradient method.

Hedged learning provides a way of deciding which model or learning method to use. We want to choose a method that does not simply converge to the best policy among our choices, since this may take a very long time. Rather, we want a method that enables us to achieve good performance in a fixed finite amount of time. Hedged learning gives us a way to choose between models and methods depending on how well they seem to be doing. Using this technique, we can thus derive performance guarantees for our algorithm while it chooses between alternative models and methods.

5.1 Regret-minimizing methods

The experts framework, and regret minimization, provides a useful tool for creating algorithms with real performance guarantees. Some of these results have been presented in prior chapters. However, as the size of the environment increases, this framework begins to break down.

In simple repeated games, it was enough to evaluate an algorithm's performance relative to the performance of the best expert assuming the opponent's string of actions remained unchanged. Within each period, we could evaluate the outcome we would have observed if a different expert had been chosen.

Now let us assume that the environment actually reacts to our action choices. For example, we might move to a different state in a stochastic game. Or, our actions

$$R_1 = \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix} \qquad R_1 = \begin{bmatrix} 2 & 3 \\ -5 & -1 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix} \qquad R_2 = \begin{bmatrix} 2 & -5 \\ 3 & -1 \end{bmatrix}$$

(a) Short-horizon PD example (b) Long-horizon PD example

Figure 5-1: Different variants of the Prisoner's Dilemna game exhibiting the need for different commitment periods.

may have convinced our opponent to change its internal state. Regardless of the reason, the environment now acts differently in response to our actions. Thus, we cannot evaluate our actions independently within each time period. For example, take the classic example of Prisoner's Dilemna. Let's assume that the opponent we face is a Tit-for-Tat player. If we play "Defect" in the current time period, then our choice action for the next time period will lead to very different outcomes compared to the situation that would have ensued had we chosen to play "Cooperate" in the current time period. In the first case, cooperating in the next time period will result in high loss. In the latter case, cooperating will the right thing to do, and will lead to mutually beneficial gains.

Fortunately, it is not hard to adapt the experts framework to deal with reactive environments. Rather than playing a chosen expert for only one time period before choosing the next expert to follow, we must now commit to following one expert's advice for multiple time steps. This commitment period is a time horizon that depends on the nature of the environment's reactivity and the reward structure of the environment. For example, in the Prisoner's Dilemna example described above, we may need a commitment period longer than two periods even though the opponent only has two internal states.

In the next section, we will refine our definitions and exact notation for the concepts just discussed, including definitions for the commitment period needed to properly estimate the value of a particular policy.

5.2 Mathematical setup

In repeated games, the standard regret minimization framework enables us to perform almost as well as the best action, if that single best action were played in every time period. Suppose we are playing using some regret-minimizing algorithm which outputs action choices $a_t \in A$ at each time period. Then our reward over T time periods is $R(T) = \sum_{t=1}^{T} r_{a_t}(t)$.

Definition 9 *Our expected regret is defined to be $R_{\max}(T) - R(T)$, where $R_{\max}(T) = \max_{a \in A} \sum_{t=1}^{T} r_a(t)$. If our algorithm randomizes over possible action choices, we also define expected regret to be $R_{\max}(T) - E[R(T)]$. The set of actions against which we compare our performance is called the comparison class.*

Both game theorists and online learning researchers have studied this framework [Fudenburg and Levine, 1995] [Freund and Schapire, 1999]. We will refer frequently to the EXP3 algorithm (and its variants) explored by Auer et al. [1995]. In the original formulation of EXP3, we choose single actions to play, but we do not get to observe the rewards we would have received if we had chosen different actions. The authors show that the performance of EXP3 exhibits a regret bound of $2\sqrt{e-1}\sqrt{TN \ln N}$. Generally speaking, these regret-minimizing algorithms hedge between possible actions by keeping a weight for each action that is updated according to the action's historical performance. The probability of playing an action is then its fraction of the total weights mixed with the uniform distribution. Intuitively, better experts perform better, get assigned higher weight, and are played more often. Sometimes these algorithms are called experts algorithms, since we can think of the actions as being recommended by a set of experts.

5.2.1 Reactive opponents

It is important to note that most of these existing methods only compare our performance against strategies that are best responses to what are often called *oblivious* or *myopic* opponents. That is, the opponent does not learn or react to our actions, and essentially plays a fixed string of actions. In this way, these existing methods are best described as $\mathcal{H}_\infty \times \mathcal{B}_0$ players. Our best response would be to play the single best-response action to the empirical distribution of the opponent's actions. Under most circumstances, however, we might expect an intelligent opponent to change their strategy as they observe our own sequence of plays. Their major benefit over other $\mathcal{H}_\infty \times \mathcal{B}_0$ players if that if the opponent turns out not to be an oblivious opponent, this approach is still able to guarantee performance that approximates the reward achieved with the best fixed action.

We can still use this type of algorithm in situations where we expect to encounter reactive opponents, but we must simply satisfy ourselves with achieving a regret-bound relative to the best we could have done against the worst-case oblivious opponent. On the one hand, this does not seem too bad. We will do as well as the best fixed action against the worst-case opponent string of actions. However, in many cases we do not face this type of opponent. We might actually do able to do better than what these worst-case algorithms guarantee.

For example, consider the game of repeated Prisoner's Dilemma. If we follow the oblivious opponent assumption, then the best choice of action would always be to "Defect." Given any fixed opponent action, the best response would always be to defect. This approach would thus miss out on the chance to earn higher rewards by cooperating with opponents such as a "Tit-for-Tat" opponent, which cooperates with us as long as we also cooperate. These opponents can be called *reactive* opponents. Mannor and Shimkin [2001] propose a super-game framework for extending the regret-minimization framework to handle such cases. In the super-game framework, we evaluate an expert's performance not based on single periods of play; instead each time we play an action or strategy, we commit to playing it for multiple time steps in order to allow the environment to react to our strategy. Pucci and Megiddo [2004] also explore this problem using a different performance metric.

5.3 Extending the experts framework

The key difference we need to make in order to account for reactive opponents is the treatment of the history of actions taken in the game. Recall that in the stochastic game model, we took $\mu_i = PD(A_i)$. As described earlier, we redefine $\mu_i : H \rightarrow A_i$, where $H = \bigcup_t H^t$ and H^t is the set of all possible histories of length t. Player i's strategy at time t is then expressed as $\mu_i(h^{t-1})$. For simplicity, we will assume $A = A_1 = A_2$. Since we cannot design agents that optimize over any possible arbitrary opponent, we focus on the largest set of opponents we can feasibly handle. One such class is the set of all opponents that use fixed length behavioral strategies.

Definition 10 *A τ-length behavioral strategy μ^τ is a mapping from all possible histories H^τ to actions $a \in A$. Let M^τ be the set of all possible τ-length behavioral strategies μ^τ.*

We note that $|M^\tau| = |A|^{|A|^{2\tau}}$. In the case where we take $H^t = H$, we could even consider learning algorithms themselves to be a possible "behavioral strategy" for playing a repeated game.

This definition of our strategy space is clearly more powerful, and allows us to define a much larger set of potential equilibria. However, when the opponent is not rational, it is no longer advantageous to find and play an equilibrium strategy. In fact, given an arbitrary opponent, the Nash equilibrium strategy may return a lower payoff than some other action. Indeed, the payoff may be even worse than the original Nash equilibrium value. Thus, regret minimization algorithms become important tools for playing competitive games in unknown settings.

Our extensions to the regret-minimization framework follow along the lines of the super-game setup described by Mannor and Shimkin [2001]. Instead of choosing actions from A, we choose behavioral strategies from M^τ. M^τ also replaces A as our comparison class, essentially forcing us to compare our performance against more complex and possibly better performing strategies. While executing $\mu^\tau \in M^\tau$ for some number of time periods λ, the agent receives reward at each time step, but does not observe the rewards he would have received had he played any of his other possible strategies. This is reasonable since the opponent may adapt differently as a particular strategy is played, causing a different cumulative outcome over λ time periods. Thus, the opponent could be an arbitrary black-box opponent or perhaps a fixed finite automaton. While the inner workings of the opponent are unobservable, we will assume the agent is able to observe the action that the opponent actually plays at each time period.

For example, we might consider an opponent whose action choices only depend on the previous τ-length history of joint actions. Thus, we can construct a Markov model of our opponent using the set of all possible τ-length histories as the state space. If our optimal policy is ergodic, we can use the mixing time of the policy as our choice of λ, since this would give us a good idea of the average rewards possible with this policy in the long run. We will usually assume that we are given λ.

Rather than playing a chosen expert for only one time period before choosing the next expert to follow, we must now commit to following one expert's advice for multiple time steps. This commitment period is a time horizon that depends on the nature of the environment's reactivity and the reward structure of the environment. For example, in the Prisoner's Dilemma example described above, we may need a commitment period longer than two periods even though the opponent only has two internal states.

For example, consider the following game structure as shown in Figure 5.1, which is an example of the Prisoner's Dilemna game. Let us assume that we keep playing the same expert, so that the played we played is the last action of this expert's commitment period choices. Then in the short-horizon version of the game, we receive higher rewards for the "Always cooperate" strategy than the "Always defect" strategy, even assuming our opponent begins the period by cooperating. Thus, a two-period horizon is enough for us to evaluate the possible strategies and conclude that "Always cooperate" is the best choice.

It even outranks the "Cooperate, then Defect (C,D)" strategy, since repeated applications of "C,D" would initialize the opponent to defect in the first period of play. This gets to a second, subtler point. Our strategies chosen in one commitment period may affect the initial state for our next period of play. Thus, we may need to assume some initial period of ramp-up time required for us to return to a favorable environmental state, such as the cooperative state of our example where we have previously cooperated and thus the opponent is inclined to cooperate on its next move. Furthermore, we need to assume that such a return to a cooperative state is even possible. We cannot allow opponents such as "Cooperate until a defection; defect forever thereafter." Playing against such an opponent, once we play "Defect" once, we can never return to a good cooperative state. Thus, we must make some assumption about the plasticity of the environment. One such possible assumption is that the environment exhibits ergodicity, which implicitly assumes that the environment can be modeled as a Markov decision process. This means that our actions will not change the environment forever, but they may have short-term consequences that must be accounted for by playing strategies for multi-period commitments. To formalize this intuition, we introduce the concept of an ϵ-commitment time, which is closely related to the standard concept of mixing time in Markov decision process.

Definition 11 *Let M be a Markov decision process that models the environment (the opponent), and let π be a policy in M such that the asymptotic average reward $V_M^\pi = \lim_{T\to\infty} V_M^\pi(i,T)$ for all i, where $V_M^\pi(i,T')$ is the average undiscounted reward of M under policy π starting at state i from time 1 to T'. The ϵ-commitment time λ_π of π is the smallest T such that for all $T' \geq T$, $|V_M^\pi(i,T') - V_M^\pi| \leq \epsilon$ for all i.*

In this chapter, we will assume that we are given a fixed value λ for each learning algorithm and opponent model, which is at least as large as the ϵ-commitment time for any ergodic policy we may wish to learn using the particular opponent model we are considering.

Thus, if we are executing a policy π learned on a particular opponent model M, then we must run the policy for at least λ time periods to properly estimate the benefit of using that policy. For example, in the Prisoner's Dilemma game shown in Figure 2-1, assuming a Tit-for-Tat opponent, we might fix $\lambda = 10$, since the average reward of playing "always cooperate" for n time periods is always within $\frac{2}{n}$ of the long-run reward. After playing this policy for 10 periods, we know that we will gain an average reward within 1/5 of the long-term average reward of 1. This is due to the fact that in the first time period, the opponent may still be playing "defect", giving us a reward of -1 for that time period. We will then receive reward of 1 in each ensuing period.

5.3.1 Evaluating all possible strategies

Given a fixed commitment length λ, we may like to be able to evaluate all possible strategies in order to choose the optimal strategy. This would entail enumerating all possible behavioral strategies over λ periods. Since the hedging algorithm will essentially randomize between strategies for us, we only need to consider deterministic behavioral strategies. However, there are still $|A|^{|A|^{2\lambda}}$ possible strategies to evaluate. Not only would this take a long time to try each possible strategy, but the regret bounds also become exceedingly weak. The expected regret after T time periods is:

$$2\sqrt{e-1}|A|^{|A|^{2\lambda}/2}|A|^{2\lambda}\sqrt{T\lambda \ln |A|},$$

Clearly this amounts to a computationally infeasible approach to this problem. In traditional MDP solution techniques, we are saved by the Markov property of the state space, which reduces the number of strategies we need to evaluate by allowing us to re-use information learned at each state. Without any assumptions about the opponent's behavior, as in the classic regret minimization framework, we cannot get such benefits.

5.4 Learning Algorithms as Experts

However, we might imagine that not all policies are useful or fruitful ones to explore, given a fixed commitment length of λ. In fact, in most cases, we probably have some rough idea about the types of policies that may be appropriate for a given domain. For example, in our Prisoner's Dilemma example, we might expect that our opponent is either a Tit-for-Tat player, an Always-Defect or Always-Cooperate player, or a "Usually Cooperate but Defect with probability p player", for example.

Given particular opponent assumptions, such as possible behavioral models, we may then be able to use a learning algorithm to estimate the model parameters based on observed history. For example, if we believe that the opponent may be Markov in the τ-length history of joint actions, we can construct a Markov model of the opponent and use an efficient learning algorithm (such as E3 from Kearns and Singh [1998]) to learn the ϵ-optimal policy in time polynomial to the number of states, $|A|^{2\tau}$. In contrast, the hedging algorithm needs to evaluate each of the exponentially large number of possible policies, namely $|A|^{|A|^{2\tau}}$ possible policies. To make this precise, we state the following lemma.

Proposition 12 *Given a model of the opponent that is Markov in the τ-length history of joint actions $\{a^i_{t-\tau}, a^{-i}_{t-\tau}, \ldots, a^i_{t-1}, a^{-i}_{t-1}\}$, and given a fixed mixing time λ, the number of actions executed by E3 and a hedging algorithm such as EXP3 in order to arrive at an ϵ-optimal policy is at most $O\left(|A|^{10\tau}\right)$ for E3, and at least $O\left(|A|^{|A|^{2\tau}}\right)$ for the hedging algorithm.*

Thus, in most cases, the learning algorithm will be much more efficient than the hedging algorithm. This is fairly obvious: whereas the hedging algorithm must try all possible policies, the learning algorithm is able to use the structure of the state space to solve for the optimal policy rather than trying each possible one.

Of course, using this method, we can no longer guarantee regret minimization over all possible policies, but as we will discuss in the following section, we can choose a subset of fixed policies against which we can compare the performance of any learning algorithms we decide to use, and we can guarantee no-regret relative to this subset of fixed policies, as well as relative to the the learning algorithms.

In some ways, using learning algorithms as experts simply off-loads the exploration from the experts framework to each individual learning algorithm. The computational savings occurs because each learning algorithm makes particular assumptions about the structure of the world and of the opponent, thus enabling each expert to learn more efficiently than hedging between all possible strategies.

5.4.1 Example

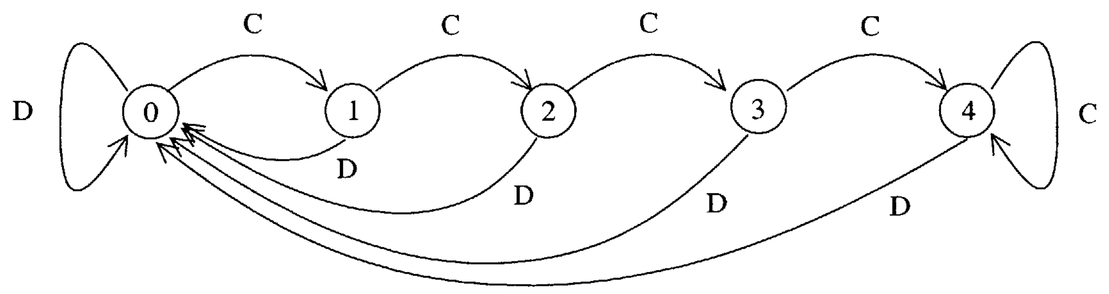


Figure 5-2: A possible opponent model with five states. Each state corresponds to the number of consecutive "Cooperate" actions we have just played.

For example, consider again the repeated Prisoner's Dilemma game. We might believe that the opponent reacts to our past level of cooperation, cooperating only when we have cooperated a consecutive number of times. If the opponent cooperates only when we have cooperated four periods in a row, then the opponent model shown in Figure 5-2 would correctly capture the opponent's state dynamics. This model is simpler than a full model using all possible 4-period histories, since it assumes that the opponent's state is completely determined by our half of the joint history. In the figure, the labeled transitions correspond to our actions, and the opponent only cooperates when it is in state 4; otherwise it defects.

To learn the optimal policy with respect this opponent model, a learning algorithm would simply have to visit all the state-action pairs and estimate the resulting reward for each possible action at each state. Since we assume that the opponent model is Markov, we can use an efficient learning algorithm such as E3.

Note that using this particular model, we can also learn the optimal policy for an opponent that cooperates if we cooperate for some given n consecutive periods, where $n \leq 4$. However, if $n \geq 5$, learning using this model will no longer result in the optimal policy. Whereas choosing the cooperate action from state 4 results in a good reward when $n \leq 4$, when $n \geq 5$ the same action results in a bad reward since the opponent will most likely play defect. The problem is that the 5-state model is no longer sufficient to capture the opponent's state dynamics, and is no longer Markov.

5.5 The Hedged Learner

Since our chosen learning algorithms will sometimes fail to output good policies, we propose to incorporate them as experts inside a hedging algorithm that hedges between a set of experts that includes our learners. This allows the hedging algorithm to switch to using the other experts if a particular learning algorithm fails. It might fail due to incorrect opponent assumptions, such as in the previous section's example, or the learning algorithm may simply be ill-suited for the particular domain, or it may fail for any other reason. The point is that we have a backup plan, and the hedging algorithm will eventually switch to using these other options.

We study two methods for adding learning experts into a regret-minimization algorithm such as Auer et al.'s EXP3. It is straightforward to extend our results to other variants of EXP3 such as EXP3.P.1, which guarantees similar bounds that hold uniformly over time and with probability one. For completeness, we provide the EXP3 algorithm here:

We are given a mixing parameter $\gamma \in (0, 1]$. We initialize the weights $w_i(i) = 1$ for $i = 1, \ldots, N$, where N is the number of experts or strategies.

For each time period $t = 1, 2, \ldots,$

1. Set

$$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^N w_j(t)} + \frac{\gamma}{N}, \text{for all} i = 1, \ldots, N.$$

2. Draw a strategy i_t randomly according to the distribution defined by the probabilities $p_1(t), \ldots, p_N(t)$, and play the action given by that strategy.

3. Receive reward $x_{i_t}(t) \in [0, 1]$.

4. For $j = 1, \ldots, N$, set

$$\begin{aligned} \hat{x}_j(t) &= \begin{cases} x_j(t)/p_j(t) & \text{if} j = i_t \\ 0 & \text{otherwise} \end{cases} \\ w_j(t+1) &= w_j(t) \exp\left(\frac{\gamma \hat{x}_j(t)}{N}\right). \end{aligned}$$

For the Hedged Learning algorithm, we are also given N fixed experts, to which we must add M learning experts. We assume that $\lambda_i = 1$ for all $i \in N$ and refer to these experts as *static experts*. These static experts are essentially the pure action strategies of the game. For all $i \in M$, we assume $\lambda_i > 1$ and note that M can also include behavioral strategies. When it is clear from context, we will often write N and M as the number of experts in the sets N and M, respectively.

- **Naive approach:** Let $\lambda_{max} = \max_i \lambda_i$. Once an expert is chosen to be followed, follow that expert for a λ_{max}-length commitment phase. At the end of each phase, scale the accumulated reward by $\frac{1}{\lambda_{max}}$ since EXP3 requires rewards to fall in the interval [0,1] and update the weights as in EXP3.

- **Hierarchical hedging:** Let E_0 denote the top-level hedging algorithm. Construct a second-level hedging algorithm E_1 composed of all the original N static strategies. Use E_1 and the learning algorithms as the $M+1$ experts that E_0 hedges between.

5.5.1 Naive approach

The naive approach may seem like an obvious first method to try. However, we will show that it is distinctly inferior to hierarchical hedging.

The naive approach proceeds as follows. Let N be the set of the original pure-action experts, and let M be the set of the learning algorithms or behavioral strategy experts. We label these experts $i = 1, 2, \ldots, M+N$. We slightly abuse notation by using $|M| = M$ and $|N| = N$ where convenient. Initialize the weights $w_i(i) = 1$ for $i = 1, \ldots, M+N$,

For each $t = 1, 2, \ldots$

1. At the beginning of each commitment phase, set

$$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{M+N} w_j(t)} + \frac{\gamma}{M+N}.$$

   Otherwise, set $p_i(t) = p_i(t-1)$.

2. If we are still in the commitment phase for expert j, set $i_t = j$. Otherwise, draw strategy i_t randomly according to the probabilities $p_1(t), \ldots, p_{M+N}(t)$. Play an action according to the strategy chosen.

3. Receive reward $x_{i_t}(t) \in [0,1]$.

4. If we have reached the end of a commitment phase for expert j, perform an update.

   For $j = 1, \ldots, M+N$, set

$$\begin{aligned} \hat{x}_j(t) &= \begin{cases} \sum_{\tau=t-\lambda_{max}+1}^{t} x_j(\tau)/p_j(t) & \text{if} j = i_t \\ 0 & \text{otherwise} \end{cases} \\ w_j(t+1) &= w_j(t)\exp\left(\frac{\gamma \hat{x}_j(t)}{\lambda_{max}(M+N)}\right). \end{aligned}$$

Using this algorithm, our regret bound becomes substantially looser than the one given for EXP3 with N pure action experts.

Theorem 13 *Suppose we have a set N of static experts, and a set M of learning experts with time horizons λ_i. Using a naive approach, we can construct an algorithm with regret bound*

$$2\sqrt{e-1}\sqrt{\lambda_{\max} T(N+M)\ln(N+M)}.$$

Proof. We run EXP3 with the $M+N$ experts, with a modification such that every expert, when chosen, is followed for a commitment phase of length $\lambda_{\max}$ before we choose a new expert. We consider each phase as one time period in the original EXP3 algorithm, and note that the accumulated rewards for an expert over a given phase falls in the interval $[0, \lambda_{\max}]$. Thus, the regret bound over $\frac{T}{\lambda_{\max}}$ phases is $2\lambda_{\max}\sqrt{e-1}\sqrt{\frac{T}{\lambda_{\max}}(N+M)\ln(N+M)}$, and the result follows immediately. □

5.5.2 Hierarchical hedging

The Naive Approach suffers from two main drawbacks, both stemming from the same issue. Because the Naive Approach follows all experts for $\lambda_{\max}$ periods, it follows the static experts for longer than necessary. Intuitively, this slows down the algorithm's adaptation rate. Furthermore, we also lose out on much of the safety benefit that comes from hedging between the pure actions. Whereas a hedging algorithm over the set of pure actions is able to guarantee that we attain at least the safety (minimax) value of the game, this is no longer true with the Naive approach since we have not included all possible $\lambda_{\max}$-length behavioral experts. Thus, each expert available to us may incur high loss when it is run for $\lambda_{\max}$ periods. Hierarchical Hedging addresses these issues.

The Hierarchical Hedging algorithm proceeds as follows:

Again, let N be the set of the original pure-action experts, and let M be the set of the learning algorithms or behavioral strategy experts. We label these experts $i = 1, 2, \ldots, M+N$. Initialize the weights $w_i(i) = 1$ for $i = 1, \ldots, M+1$, and initialize a second set of weights $v_i(i) = 1$ for $i = 1, \ldots, N$.

For each $t = 1, \lambda_{max} + 1, 2\lambda_{max} + 1, \ldots$

1. At the beginning of each commitment phase of length λ_{max}, set

$$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{M+1} w_j(t)} + \frac{\gamma}{M+N}.$$

2. Draw strategy i_t randomly according to the probabilities $p_1(t), \ldots, p_{M+1}(t)$. If $i_t \in M$, follow 3a. Otherwise, if $i_t \notin M$, follow 3b.

3a. For each time period $\tau = t, t+1, \ldots, t+\lambda_{max} - 1$, play an action according to the chosen strategy i_t and receive reward $x_{i_t}(\tau) \in [0, 1]$.

3b. For each time period $\tau = t, t+1, \ldots, t+\lambda_{max} - 1$,

1. Set

$$q_i(\tau) = (1-\gamma)\frac{v_i(\tau)}{\sum_{j=1}^{N} v_j(\tau)} + \frac{\gamma}{N}, \text{for all} i = 1, \ldots, N.$$

2. Draw and play an action k_τ randomly according to the distribution defined by the probabilities $q_1(\tau), \dots, q_N(\tau)$.

3. Receive reward $x_{k_\tau}(\tau) \in [0,1]$.

4. For $j = 1, \dots, N$, set

$$\begin{aligned} \hat{x}_j(\tau) &= \begin{cases} x_j(\tau)/q_j(\tau) & \text{if} j = i_\tau \\ 0 & \text{otherwise} \end{cases} \\ v_j(\tau+1) &= v_j(\tau) \exp\left(\frac{\gamma \hat{x}_j(\tau)}{N}\right). \end{aligned}$$

4. Since we have reached the end of a commitment phase for expert i_t, perform an update.

For $j = 1, \dots, M+1$, set

$$\begin{aligned} \hat{x}_j(t) &= \begin{cases} \sum_{\tau=t-\lambda_{max}+1}^{t} x_j(\tau)/p_j(t) & \text{if} j = i_t \\ 0 & \text{otherwise} \end{cases} \\ w_j(t+\lambda_{max}) &= w_j(t) \exp\left(\frac{\gamma \hat{x}_j(t)}{\lambda_{max}(M+1)}\right). \end{aligned}$$

Theorem 14 *Suppose we have a set N of static experts, and a set M of learning experts with time horizons λ_i, $\max_i \lambda_i > |N|$. We can devise an algorithm with regret bound:*

$$\begin{aligned} & 2\sqrt{e-1}\sqrt{TN \ln N} \\ + \; & 2\sqrt{e-1}\sqrt{\lambda_{\max} T(M+1)\ln(M+1)} \quad . \end{aligned}$$

This upper bound on the expected regret improves upon the Naive Approach bound as long as

$$\lambda_{\max} \geq \frac{\sqrt{\ln N}}{\sqrt{\ln(M+N)} - \sqrt{\ln(M+1)}}.$$

In practice, we will often use only one or two learning algorithms as experts, so M is small. For $M = 1$, the bound would thus look like:

$$2.63\sqrt{TN \ln N} + 3.10\sqrt{\lambda_{\max} T}.$$

However, we note that these are simply upper bounds on regret. In Section 5.5.3, we will compare actual performance of these two methods in a some test domains.

Proof. Using the bounds shown to be achieved by EXP3, our top-level hedging algorithm E_0 achieves performance

$$R_{E_0} \geq \max_{i \in M + \{E_1\}} R_i - 2\sqrt{e-1}\sqrt{T(M+1)\ln(M+1)}.$$

Now consider each of the $|M|+1$ experts. The $|M|$ learning experts do not suffer additional regret since they are not running another copy of EXP3. The expert E_1 is running a hedging algorithm over $|N|$ static experts, and thus achieves performance bounded by

$$R_{E_1} \geq \max_{j \in N} R_j - 2\sqrt{e-1}\sqrt{\lambda_{\max} T N \ln N}.$$

Combining this with the above, we see that

$$\begin{aligned} R_{E_0} \;\geq\; & \max\nolimits_{i \in M+N} R_i \\ & -2\sqrt{e-1}\sqrt{TN \ln N} \\ & -2\sqrt{e-1}\sqrt{\lambda_{\max} T(M+1)\ln(M+1)}. \;\square \end{aligned}$$

Proposition 15 *The Hierarchical Hedging algorithm will attain at least close to the safety value of the single-shot game.*

Proof. From an argument similar to [Freund and Schapire, 1999], we know that the second-level expert E_1 will attain at least the safety value (or minimax) value of the single-shot game. Since the performance of the overall algorithm E_0 is bounded close to the performance of any of the experts, including E_1, the Hierarchical Hedger E_0 must also attain close to the safety value of the game. □

As desired, hierarchical hedging is an improvement over the naive approach since: (1) it no longer needs to play every expert for $\lambda_{\max}$-length commitment phases and thus should adapt faster, and (2) it preserves the original comparison class by avoiding modifications to the original experts, allowing us to achieve at least the safety value of the game.

Remark. It is also possible to speed up the adaptation of these hedged learners by playing each expert i for only λ_i time periods, weighting the cumulative rewards received during this phase by $1/\lambda_i$, and using this average reward to update the weights. Applied to the hierarchical hedger, we would play each learning algorithm i for λ_i-length phases and the second-level hedging algorithm E_1 for N-length phases. In practice, this often results is some performance gains.

5.5.3 Practical comparisons

We can verify the practical benefit of hierarchical hedging with a simple example. We consider the repeated game of Matching Pennies, shown in Figure 2-1. Assume that the opponent is playing a hedging algorithm that hedges between playing "Heads" and "Tails" every time period. This is close to a worst-case scenario since the opponent will be adapting to us very quickly.

We run each method for 200,000 time periods. The Hierarchical Hedger consists of 9 single-period experts grouped inside E_1 and one 500-period expert. The Naive Hedger runs all the experts for 500 periods each. The results are given in Table 5.1, along with the expected regret upper bounds we derived in the previous section. As expected, the hierarchical hedger achieves much better actual performance in terms of cumulative reward over time, and also achieves a lower expected regret. However, the regret for the naive approach is surprisingly low given that its performance is so poor. This is due to a difference in the comparison classes that the methods use. In the naive approach, our performance is compared to experts that choose to play a single action for 500 time periods, rather than for a single time period. Any single action, played for a long enough interval against an adaptive opponent, is a poor choice in the game of matching pennies. The opponent simply has to adapt and play its best response to our action, which we are then stuck with for the rest of the interval. Thus the expected rewards for any of the experts in the naive approach's comparison class is rather poor. For example, the expected reward for the "Heads" expert is -98,582. This explains why our expected regret is small, even though we have such high cumulative losses; we are comparing our performance against a set of

Table 5.1: Comparison of the performance of the different methods for structuring the hedged learner.

	Regret Bound	Actual Expected Regret	Actual Performance
Naive	125,801	34,761	-96,154
Hierarchical	36,609	29,661	-8,996

poor strategies!

5.6 Experimental Results

Since the worst-case bounds we derived in the previous section may actually be quite loose, we now present some experimental results using this approach of hedged learning. We consider the repeated Prisoner's Dilemma game, and we first assume that the unknown opponent is a "Tit-for-Ten-Tats" opponent. That is, the opponent will only cooperate once we have cooperated for ten time periods in a row.

5.6.1 Hedging between models

We use a variety of different opponent models with simple learning algorithms, pure hedging algorithms that only hedge between static experts, and hedged learning algorithms that combine learning algorithms with static experts. First, we note that larger opponent models are able to capture a larger number of potential opponent state dynamics, but require both a longer commitment phase λ and a larger number of iterations before a learning algorithm can estimate the model parameters and solve for the optimal policy. For example, Figure 5-3 shows the performance of three different n-state learners, with $n = 4, 10, 30$. As discussed earlier in Section 5.4, the 4-state learner is unable to capture the opponent's state dynamics and thus learns an "optimal" policy of defecting at every state. This results in an average reward of zero per time step. On the other hand, the 10-state and 30-state learners lose some rewards while they are exploring and learning the parameters of their opponent mod-

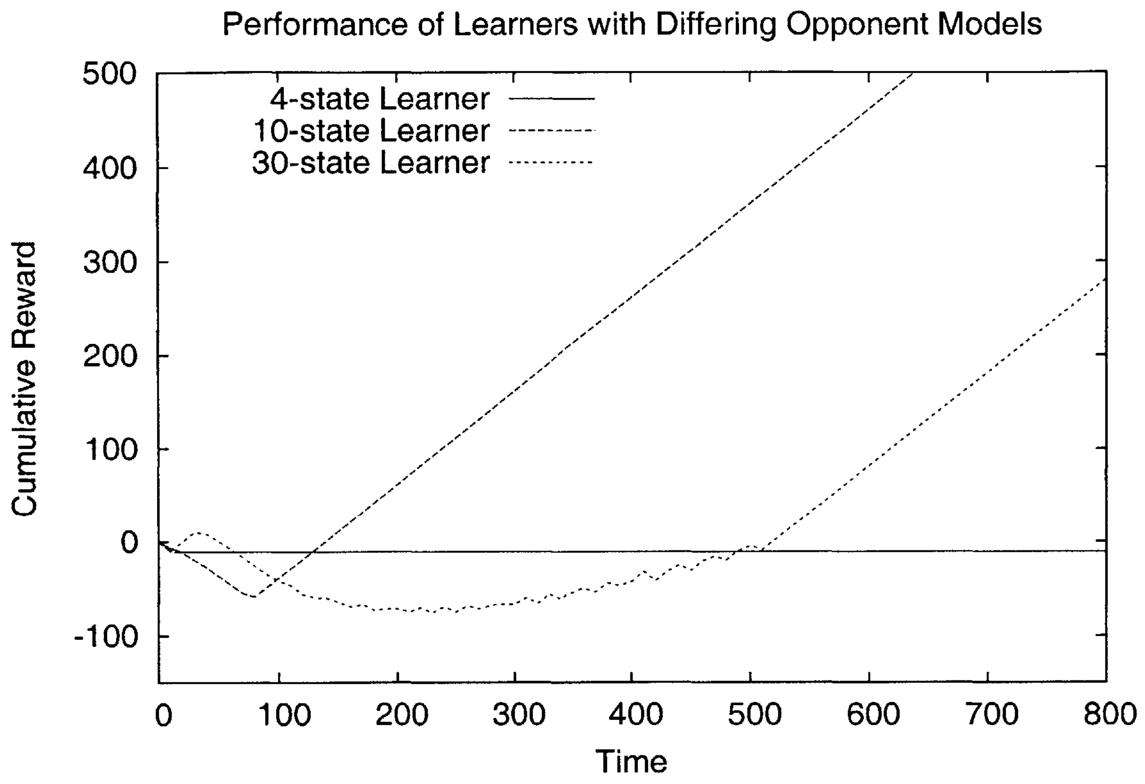


Figure 5-3: This graph shows the performance of learning algorithms against a Tit-for-Ten-Tats opponent. As the opponent model grows in size, it takes longer for the learning algorithm to decide on an optimal policy.

els, but then gain an average reward of 1 after they have found the optimal policy of always cooperating.

Figure 5-4 shows the performance of various learning and hedging algorithms. The "1-period experts" hedging algorithm hedges between single periods of cooperating and defecting. This myopic algorithm is unable to learn the cooperative outcome and thus ends up achieving the single-shot Nash equilibrium value of 0. It assigns a very high weight to the Defect expert. On the other hand, the "25-period experts" hedging algorithm switches between two experts which either cooperate or defect for all possible 25-period histories. This algorithm realizes that the "always cooperate" expert attains higher reward and thus eventually plays Cooperate with probability approaching 1. The hedged 10-state learner is also able to achieve the cooperative

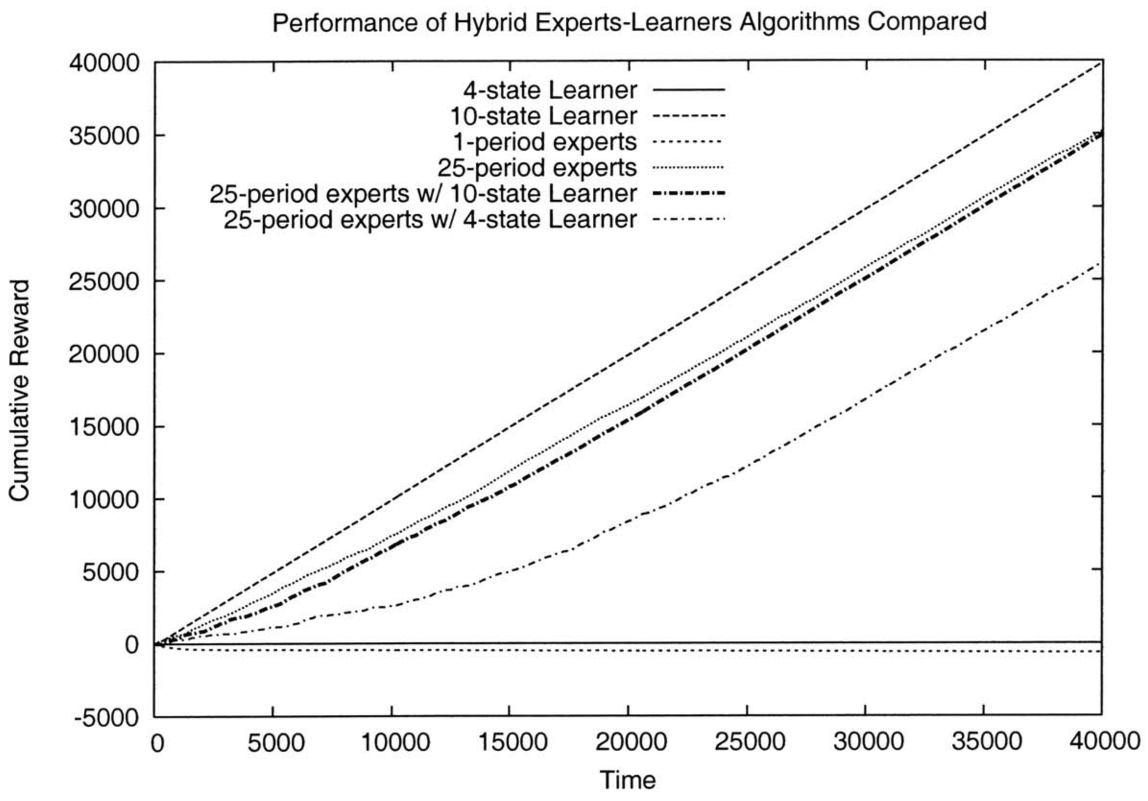


Figure 5-4: This chart shows the performance of different learning, hedging, and hedging learning algorithms in a game of repeated prisoner's dilemma against a Tit-for-Ten-Tats opponent.

outcome. It achieves cumulative reward only slightly lower than the un-hedged 10-state learner, since it quickly realizes that the "always cooperate" policy and the learned optimal policy both return higher rewards than the "always defect" policy.

One main benefit of the hedged learning approach becomes evident when we observe the performance of the hedged 4-state learner. Even though the 4-state model is unable to capture the state dynamics and the learning algorithm thus fails to learn the cooperative policy, the hedged 4-state learner is able to achieve average rewards of 1 as it assigns larger and larger weight to the "always cooperate" expert and learns to ignore the recommendations of the failed learning expert. We have wisely hedged our bets between the available experts and avoided placing all our bets on the learning algorithm.

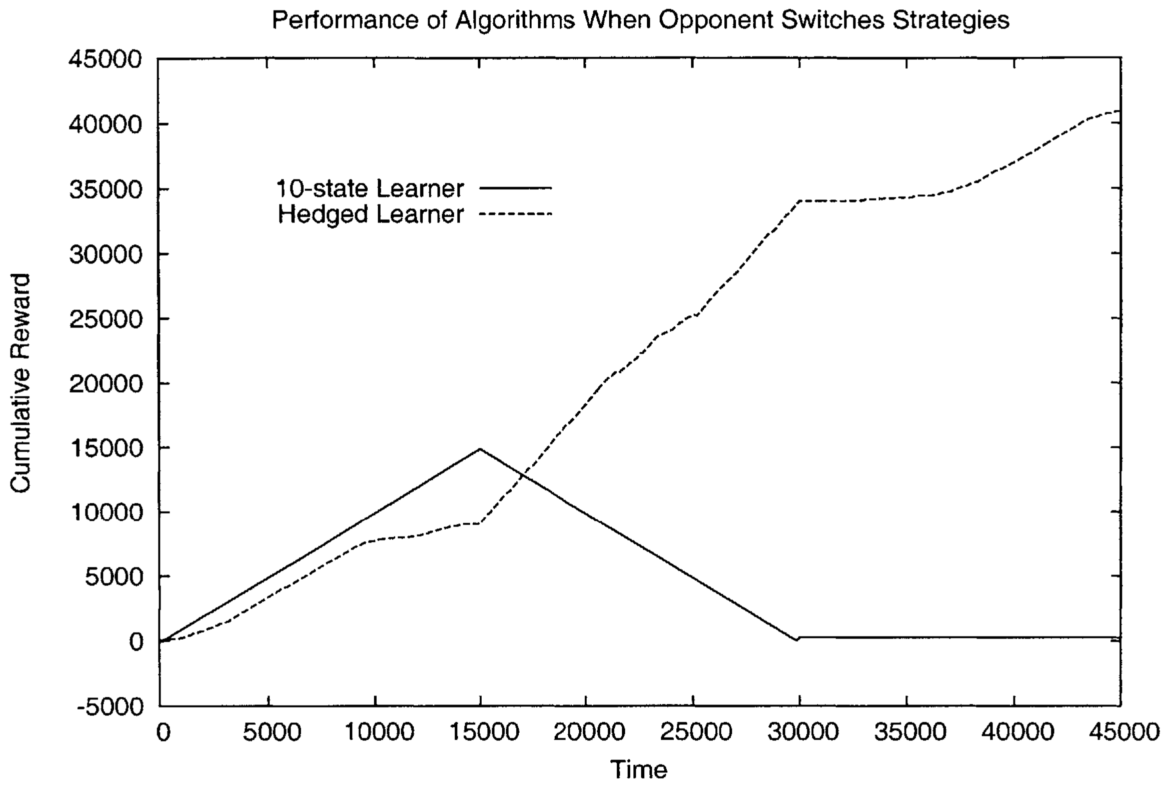


Figure 5-5: In these trials, the opponent switches strategy every 15,000 time periods. It switches between playing Tit-for-Ten-Tats ("Cooperate for 10 Cooperates") and "Cooperate for 10 Defects". While the modeler becomes confused with each switch, the hedging learner is able to adapt as the opponent changes and gain higher cumulative rewards.

5.6.2 Non-stationary environments

Another major benefit of using hedged learners occurs when the environment is non-stationary. For example, assume that the opponent switches between playing Tit-for-Ten-Tats ("Cooperate for 10 Cooperates") and "Cooperate for 10 Defects" every 15,000 time periods. While the unhedged learner becomes confused with each switch, the hedged learner is able to adapt as the opponent changes and gains higher cumulative rewards (Figure 5-5). Note that when the opponent does its first switch, the unhedged learner continues to use its cooperative policy, which was optimal in the first 15,000 periods but now returns negative average reward. In contrast, the hedged learner is able to quickly adapt to the new environment and play a primarily defecting

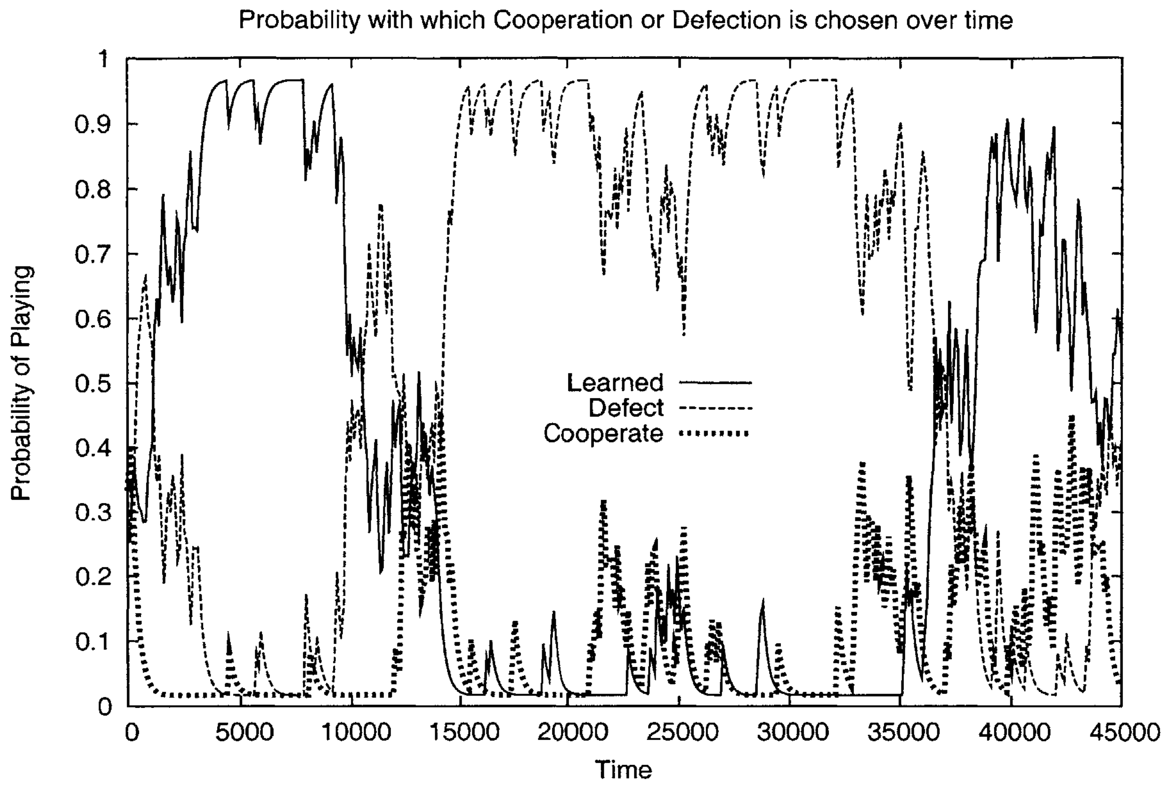


Figure 5-6: Graph showing the probability with which the weighted hedger plays either a cooperating strategy or a defecting strategy against the switching opponent over time.

string of actions. Figure 5-6 shows how the hedging algorithm is able to change the probabilities with which it plays each expert as the environment changes, i.e. when the opponent switches strategies.

5.7 Approximately optimal solutions to MDPs

Lets focus on one particular expert within this hedged learning framework for a moment. The scenario is as follows. The domain is a stochastic game, where we can observe states of the world and where we might also have features that provide us information about the internal states of the opponent. These states provide us with potentially a good enough description of the model to learn a proficient reactive policy that simply maps observed states to action choices. Thus, we could simply employ a standard MDP learner to find an optimal or near-optimal policy. However, we might also be unsure whether our model is actually correct. In case it is not correct, we might incur significant losses trying to find and follow a policy that is near-optimal for an incorrect model. By combining the MDP learner with other experts in an experts framework, we can guarantee minimum performance levels while also guaranteeing polynomial-time convergence to a near-optimal MDP solution.

The first polynomial-time probably near-optimal algorithm for solving general MDPs was given by Kearns and Singh [Kearns and Singh, 1998]. As the authors state, they make no attempt to optimize the bounds they give; their goal was simply to show that polynomial-time bounds were possible.

The original bound given by Kearns and Singh guarantee a probably near-optimal policy in an unknown MDP in time bounded by

$$O\left(N^3 T (G^T_{max}/\epsilon)^2 \log(N/\delta) m_{known}\right),$$

where T is the mixing time, N is the number of states in the MDP, G_{max} is the maximum reward at any state, m_{known} is the number of times each action from a given state must be executed before we declare the state to be "known", δ is the probability with which our bound fails, and ϵ is deviation from the optimal policy value. m_{known} is of the order $O\left(((NTG^T_{max})/\epsilon)^4 Var_{max} \log(1/\delta)\right)$.

There are some clear benefits when we plug E3 into the experts framework. Since E3 is model-based, it does not rely on on-policy exploration of the state space. Thus, even if we are not executing E3 in our experts algorithm, we can still update the MDP we are trying to learn within E3. Each action we take may land us in a new state, which we should count as a state visit within E3. We can update the state's transition and reward statistics accordingly.

There is only one major detail to which we need to pay careful attention. E3 relies on an "attempted exploration" step as a key component of the algorithm. This is a T-step attempt to find a state that is not yet known. Thus, our τ-horizon cannot be shorter than T period. But, in fact, the mixing time T is a natural choice for τ in this setup.

5.8 Extensions

As we've seen in this chapter, hedged learning is able to incorporate various possible opponent models as well as various possible fixed strategies into its hedging framework. It is thus able to benefit from the efficiencies of using learning algorithms to learn optimal policies over these models, while ensuring that it has the option of falling back on the fixed strategies in case the learning algorithms fail to output any good policies. Even when the opponent switches strategies during play, hedged learning is able to adapt to its changing environment and switch to using the expert best adapted to the new opponent strategy.

We note that all of our results still hold for > 2 players, since we only require observations of our own reward and the history of actions. Furthermore, we can also extend these techniques for use in stochastic games. For example, we have applied this technique to the a simplified game of soccer, once again adapted to the grid-world environment. The setup and thus the results are very similar, since the hedged learning algorithm only relies on the given experts and learning algorithms to deal with the added external state. However, given large state spaces, the algorithm becomes less efficient, since learning takes longer, and there are ever more behavioral strategies to consider. The problem is no different from increasing the length of the behavioral strategies that we are willing to consider for our opponent models.

One of the main thrusts for future work will be the development of rich opponent models whose structural properties allow us to increase the efficiency and performance of our hedged learning algorithm by giving us the ability to reuse information across multiple experts. Rather than the current situation where we must in essence evaluate each expert separately, we might be able to design experts such that an evaluation of one expert also provides us with some information about the other experts' expected performance if they were to be evaluated on the same trial.

Chapter 6

Mobilized ad-hoc networking

So far we have developed a large suite of methods for dealing with different types of multi-agent learning problems. Most of our experiments have focused on small toy domains. In this chapter, we will apply our algorithms to a large, realistic domain: mobilized ad-hoc networks. Here we are confronted with all of the difficulties that have been raised thus far: a large domain with many agents, partially observability due to the size of the domain, and a global reward signal that is given to all the agents regardless of individual merit. One of the only simplifications we can take comfort in is the fact that the network will be assumed to be cooperative. That is, all of the nodes are assumed to have the same overall goal in mind and will act cooperatively to achieve this goal. One could easily imagine an extension of this domain in which there might be an adversary that attempts to jam the network, but we do not consider that possibility here.

6.1 Domain overview

Our mobilized ad-hoc network consists of one or more source nodes, one or more receiver nodes, and a number of other wireless nodes within a constrained area. All nodes are independently initialized according to a uniform distribution over this area. The sources generate packets at a constant rate and move according to a random way-point model. The aim of all nodes other than the sources and receivers is to transmit the maximum possible number of packets from the sources to the receivers. Some of these nodes can move so as to aid transmission, while the rest are stationary.

Performance is measured by the proportion of packets successfully transmitted to the receivers. When inter-node link capacities are limited and buffer sizes are finite, packet drops may occur due to buffer overflow, thereby decreasing network performance. When inter-node link capacities are sufficiently large compared to the source packet generation rates, an equivalent performance metric is the average proportion of sources connected to the receivers over time.

6.1.1 Performance benefits of mobility

Clearly, the mobility of our agent nodes increases the performance of our ad-hoc network by allowing these nodes to move to locations where they are most needed. In this section, we attempt to quantify this performance benefit with respect to several network parameters.

The packet transmission success probability achievable in an ad-hoc network depends heavily on various parameters including the transmission range r, the number of network nodes n, and the proportion of mobile nodes m. Alternatively, we may consider what values of these parameters are sufficient to achieve a desired success probability. Our following result illustrates these relationships and gives some sense of the potential benefits of having controllable mobile nodes in an ad-hoc network. It builds upon a theorem by Gupta and Kumar [1998] that sets a bound for the minimum (or critical) range necessary for n nodes in a given area to be fully connected. They state that for the case of a disk of area A and n approaching infinity, the critical range for the network is $r_n = \sqrt{\frac{A(\log n+\gamma_n)}{\pi n}}$, where γ_n is a function of n that grows arbitrarily slowly to infinity as $n \to \infty$. We extend this result to the case of partiallly mobile networks.

Proposition 16 *Let r_n be the minimum, or critical, range needed for n nodes independently and uniformly distributed on a given area to form a fully connected network. If a proportion $m = \frac{k-1}{k}$, k an integer, of the nodes are mobile, then a transmission* *range $r = \frac{r_n}{\sqrt{k}}$ is sufficient to make it possible for the mobile nodes to move to form a fully connected network. If the range r is fixed at r_n, with $m = \frac{k-1}{k}$, k an integer, then a total number $\frac{n}{k}$ of nodes suffices for full connectivity.*

Proof. If a proportion $m = \frac{k-1}{k}$ of the nodes in a n node network being mobile, these mn nodes can essentially move around to increase the effective transmission ranges of the stationary nodes. For each fixed node, we have $k-1$ mobile nodes. Thus, each stationary node is able to form one link to neighboring node over a distance of kr_n using a set of $k-1$ mobile nodes as relays. Since a fully connected network of n nodes requires only $n-1$ links, each link can be allocated a set of $\frac{(k-1)n}{k(\frac{n}{k}-1)} > k-1$ mobile relay nodes, allowing each stationary node to transmit a full distance of kr_n rather than r_n. Since $n' = \frac{n}{k}$, from Gupta and Kumar's theorem we know that if all the nodes were stationary, then the critical range for n' nodes would be $r_{n'} = r_{\frac{n}{k}} = \sqrt{\frac{A \log n' + \gamma_{n'}}{\pi n'}} < \sqrt{\frac{A \log n + \gamma_n}{\pi \frac{n}{k}}} = r_n\sqrt{k}$. However, since $(k-1)n$ nodes are now mobile, making the transmission range k times the original range, we only need a range of $r = \frac{r_n}{k}$ for each node; thus a sufficient range for fully connectivity of the partially mobile network is $r = \frac{r_n}{\sqrt{k}}$.

Using the same reasoning, it is easy to show the second part of the theorem. Let us fix the range of the n nodes in the network to be $r_n = \sqrt{\frac{A(\log n + \gamma_n)}{\pi n}}$. We need to show that with $n' = \frac{n}{k}$ total nodes, of which mn' are mobile, each with range r_n, we can form a fully connected network. In this case, we only have $n'' = \frac{n}{k^2}$ stationary nodes. Similar to the above, we know that the critical range for a network with n'' nodes is $\sqrt{\frac{A(\log \frac{n}{k^2} + \gamma_{n/k^2})}{\pi \frac{n}{k^2}}} < r_n k$. Since mn' nodes are mobile, $m = \frac{k-1}{k}$, we actually only need a range of $\frac{r_n k}{k} = r_n$. □

This result shows that as the proportion of mobile nodes increases, the transmission range r needed for full connectivity decreases for given n (or the minimum value of n required given r decreases). These are loose upper bounds since they allow for up to all links of the fully connected network being greater than $\frac{k-1}{k}$ of the maximum link length.

For more realistic settings with smaller fixed n, we can obtain corresponding empirical results for the minimum range r necessary for achieving nearly full connectivity

(> 95%). We ran some trials with n nodes, $25 \leq n \leq 400$, independently initialized according to the uniform distribution on a unit square. We find that the approximation $r'_n = \sqrt{\frac{A \log n}{\pi n}} \approx r_n$ gives a good estimate of the critical range to within a small decreasing factor. We also calculated the mean proportion h of links that were greater than half the critical range.

n	critical range	mean proportion h	std. dev.
25	$2.0r'_n$	0.2735	0.1233
50	$1.7r'_n$	0.3265	0.0785
100	$1.7r'_n$	0.2353	0.0647
200	$1.6r'_n$	0.1992	0.0390
400	$1.6r'_n$	0.1588	0.0258

Noting that a substantial proportion of the links are less than half the critical range leads to the conclusion that the bounds given above are quite loose, i.e. the necessary values for each parameter r, n and m, given fixed values for the other two parameters are lower than the sufficient values given by Theorem 16. As such, we would expect potentially better results under conditions of global knowledge and rapid optimal deployment of mobile nodes.

In most realistic settings, nodes only possess local knowledge and movement speed is limited. Thus it becomes harder to reach the potential optimal network performance. Nevertheless, we can develop learning algorithms that perform well given these constraints. In the remainder of this chapter, we develop such learning algorithms for movement and routing, and compare their performance against both non-learning and centralized algorithms, where performance is measured by the proportion of packets successfully transmitted under various network scenarios.

6.2 The routing problem

To optimize the performance of the ad-hoc network, we need to design good control algorithms for the support nodes. The nodes will need to adapt to changing conditions and communication patterns using intelligent routing and movement algorithms. We focus on the routing problem in this section.

Q-routing [Boyan and Littman, 1994] is an adaptive packet routing protocol for static networks based on the Q-learning algorithm, which we adapt for use in mobile ad-hoc networks. The algorithm allows a network to continuously adapt to congestion or link failure by choosing routes that require the least delivery time. When a route becomes congested or fails, Q-routing learns to avoid that route and uses an alternate path. Due to its adaptive nature, we might expect that Q-routing would also work well in the mobile ad-hoc setting.

Q-routing is a direct application of Watkins' Q-learning [Watkins, 1989] to the packet routing problem. Q-routing is a distributed routing scheme where each node in the network runs its own copy of the Q-routing algorithm. A node x faces the task of choosing the next hop for a packet destined for some receiver node d. Using Q-routing, it learns the expected delivery times to d for each possible next hop y, where each possible next hop y is a neighbor node connected to x by a network link. Formally, Q-routing keeps Q-tables Q^x for each node x and updates these tables at each time period t as follows:

$$Q_t^x(d,y) = (1-\alpha)Q_{t-1}^x(d,y) + \alpha(b_t^x + \min_z Q_{t-1}^y(d,z)),$$

where $0 < \alpha < 1$ is parameter that controls the learning rate, and b_t is the time the current packet spent on the buffer or queue at node x before being sent off at time period t.

Q-learning estimates the *value* or *cost*, V, associated with each state d, with $V = \min_z Q^x(d,z)$. In our case, the value of a state is the estimated time for delivery of a packet from the current node x to destination d via node z. Once the nodes have learned the values associated with each state-action pair, they simply execute a greedy policy to behave optimally. When a node receives a packet for destination d, it sends the packet to the neighbor y with the lowest estimated delivery time $Q^x(d,y)$. In experimental trials, Boyan and Littman found that fast learning rates (around 0.5) worked well since it is beneficial for the network to adapt quickly.

6.3 The movement problem

The problem of learning a good movement policy is much more difficult. We wish to again apply reinforcement learning techniques to this problem. First of all, the problem of partial observability is much more pronounced than in the packet routing problem. For example, when the network is in a disconnected state, information about the global network topology is impossible to collect but would be important for determining movements that would optimize network connectivity. Moreover, the local observation space could still be quite large, depending on the observed variables we choose to encode. Secondly, the choice of action space is unclear. At the most basic level, the agents could move in any direction at a given speed. We will limit the action choices by designing more complex actions that incorporate domain knowledge. This section briefly outlines our application of Q-learning to learn a reasonable movement policy despite the fact that Q-learning generally fails in POMDPs.

We proceed by using the observation space as our "state space". This can potentially lead to problems of aliasing, but we choose our observed variables carefully in order to try to avoid this pitfall. Since the nodes communicate using a shared wireless medium, a node can "sniff" packets sent by neighbors to destinations other than itself. Thus, a node can detect the presence of neighbor nodes and their connections, even if it is not involved in any of these connections itself. Moreover, since the receiver nodes send back acknowledgement packets along these connections, our agent node can also collect statistics about these source-destination paths by sniffing these acknowledgement packets. Each agent node's observation space thus includes the number of neighbors, the number of connections it is currently holding, the number of nearby connections, and the maximum and minimum average hop lengths of these source-destination paths.

In order to constrain the learning process, we incorporate some domain knowledge into our design of an appropriate action space. For example, there is little need to train the nodes to avoid obstacles along their desired path of movement. We can pre-program the nodes with the necessary algorithm to do this. Learning is focused on higher-level action selection that is difficult to pre-program effectively. A subset of our action space is given in the table below. Many of these actions could take multiple time periods to complete. We get around this problem by allowing the agents (nodes) to choose to either continue or change an action during each time period.

Action	Description
stay	Stay put; don't change position.
direction	Head in a particular direction.
plug	Searches for the sparsest path and attempts to fill in the largest gap along that path.
leave	Leaves a path and becomes available for other actions.
circle	Circles around a node in search of more connections. Attempts to retain connection to the source around which it is circling.
lead	Leads other mobile agents in search of more connections. Attempts to stay within range of its followers.
follow	Identifies and follows a leader node, while maintaining connection to previous hop.
center	Moves to the centroid of the neighbors to which it is connected.
explore	Random exploration.

Finally, the reward given to the nodes during each time period corresponds to the percentage of successful transmissions during that time period, which is available since we are conducting this training off-line. During online execution of the ad-hoc network, this network performance measurement would not be available since there would be no trainer that has access to the global state of the network. With these assumptions, we can construct a standard Q-learning algorithm that tries to learn good movement policies as the agents interact in simulation.

6.4 Application of our methods

6.4.1 Routing

Adapting Q-routing to the mobile ad-hoc network routing domain is fairly straightforward. We use the same Q-value updates as before:

$$Q_t^x(d,y) = (1-\alpha)Q_{t-1}^x(d,y) + \alpha(b_t^x + \min_z Q_{t-1}^y(d,z)).$$

The main difference is that we now interpret the variables differently. Neighbor nodes are now defined as the nodes within transmission range. Unlike in the static networking case, here neighbor nodes y may appear and disappear quite frequently due to node mobility. When a node y moves out of range, we set the estimated delivery time to d via y to ∞; i.e., $Q^x(d,y) = \infty$. When a node y moves into range, we optimistically set the estimated time to 0; i.e., $Q^x(d,y) = 0$. This optimistic bias encourages exploration. That is, node x will always try sending packets via a node y that has just come into range. If this action results in a high estimated delivery time, then node x will quickly revert to its original behavior since $Q^x(d,y)$ will quickly be updated to its true value. On the other hand, if this action results in a good delivery time, then node x will continue to send packets via node y.

6.4.2 Movement

The movement problem represents a much harder problem. Our goal is to learn a movement policy that maps local state information, possibly combined with some internal state information, to distributions over actions. Difficulties arise from several different fronts. First of all, the agents only receive global learning signal that tells them the overall performance of the ad-hoc network. Since the agents only have a local representation for the state space, it is difficult for them to deduce the credit assignment that is due to them.

This local knowledge also leads to the second main problem, which is that the agents only receive partial observations of their environment. Without being able to see the global state of the network (including the locations of all the other nodes), the agents cannot make an optimal decision regarding their own next moves. Thus, they must base their decisions on the local information that is available. In some cases, it may be beneficial to use memory to capture some more information about unobserved states. Furthermore, since nodes are often connected to many other nodes, they can piggyback some global state information on the data packets that are being routed through the network.

In the remainder of this section, we discuss various strategies that we can employ to overcome these difficulties. Our discussion will begin from the most basic algorithm, Q-learning. From there, we will explore more complex learning methods that begin to account for the complex nature of the domain we are studying.

6.4.3 Basic Q-learning

We have already described the basic Q-learning algorithm and applied it to the mobilized ad-hoc network packet routing problem. To apply Q-learning to the movement problem, we need to first decide on an appropriate state space. The simplest state space to use would be the space of local observations. Using this state space, we can hope to learn a purely reactive policy that maps local observations about the agent's state to action choices.

There are still many different possible features of the local observation space that we might wish to include in the state space. The four features we choose for the state space are: (1) the number of source nodes to which the node is connected, (2) whether it is connected to a receiver node, (3) a discretized measure of the number of neighbors in its transmission range, and (4) a discretized measure of the number of connections being routed within its transmission range.

6.4.4 Filtered learning

As described in Chapter 4, we can easily add our filtering technique to this Q-learning implementation. Rather than simply training using the supplied global reward signal, we instead train using the filtered signal. In Chapter 4, we saw that this led to performance gains in the simplified mobilized ad-hoc networking problem. The gains in this full implementation are much less clear. A model-based version of the filtered learner appear to perform better, but it is also inconsistent. Part of the problem is the inherent high variability of this system, since the system's performance depends in large part on the movement trajectories of the source nodes.

6.4.5 Gradient methods

One method for dealing with the partially observable nature of this domain is to use methods that were specifically designed for POMDPs. Since we have no hope of solving this large POMDP exactly, we attempt to use approximate, locally-optimal solutions such as approximate policy gradient estimation. Here we use the simple REINFORCE algorithm proposed by Williams [1992]. We adapt REINFORCE to our current domain by breaking up the simulations into fixed length trials, and using these trials to estimate the policy gradient. This method is quite slow, and its success depends very much on the initialization of the policy.

6.4.6 Hedged learning

Finally, we use the techniques we developed in Chapter 5 to construct a hybrid algorithm for learning good movement policies in this domain. This type of technique is particularly useful if we need to have the agents adapt to their environments online, since we can include fixed policies that have been trained on various types of simulated scenarios. These fixed policies would appear as static strategies within the hedged learning framework. We could also include other learning algorithms as experts, which could act as a safeguard in case none of the pre-trained policies results in a good level of performance.

In particular, we use 3 fixed policies: (1) the stationary policy where nodes do not move, (2) a hand-coded policy, and (3) a policy trained using the model-based learner with the observation state space described above. We also include an untrained model-based learning algorithm as one of the experts.

6.5 Empirical results

6.5.1 Empirical results for routing

Our empirical results give an indication of the power of using adaptive learning techniques in designing movement and routing algorithms for mobilized ad-hoc networks. The setup is described in Section 6.1. There are source, receiver, and support nodes in a square grid, usually 30x30 in size. Each node has a transmission range of 6. The support nodes may either be fixed stationary nodes or mobilized agent nodes. There are two time scales: one for transmission and one for movement. During each movement time step, the node can choose one of its movement actions and perform 10 packet transmissions. Each packet transmission is the result of a Q-routing decision and update. Sources generate packets every two transmission time steps, and the number of packets received by a node in any time period is only limited by the node's buffer size.

To evaluate the performance of Q-routing, we implement a global-knowledge routing policy that is given information about the receiver location. This is done for comparison purposes only; in reality nodes generally do not have access to such information. With this information, nodes can route each packet towards the correct destination by forwarding the packet to a neighboring node that is closest to being in the direction of the receiver. We call this our *directional routing* policy. Specifically, our implementation forwards a packet to the neighbor that is closest to being in the direction of the receiver, up to a maximum deviation of 90 degrees. If no such neighbor exists, then the packet is dropped.

We compared Q-routing with directional routing under a variety of different scenarios. In almost all cases, Q-routing performs better than directional routing. Especially when the nodes are limited by small buffer sizes, Q-routing performs significantly better. Results for a typical set of network scenarios are shown in Figure 6-1.

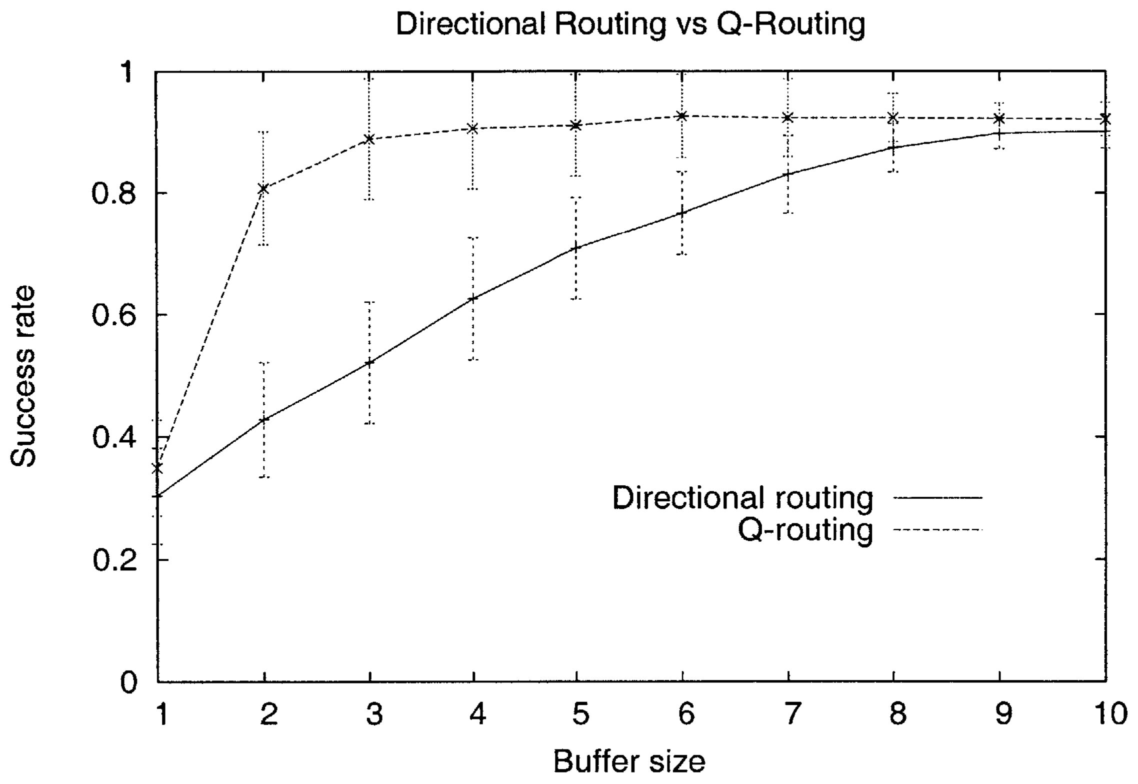


Figure 6-1: A comparison of directional routing vs Q-routing in a network with 10 sources, 15 mobile agents, and one receiver. Simulations were run over 20 initialization positions and 5 source movement scenarios for each different initialization. For each buffer size, averages over all of these trials are depicted, with error bars denoting one standard deviation.

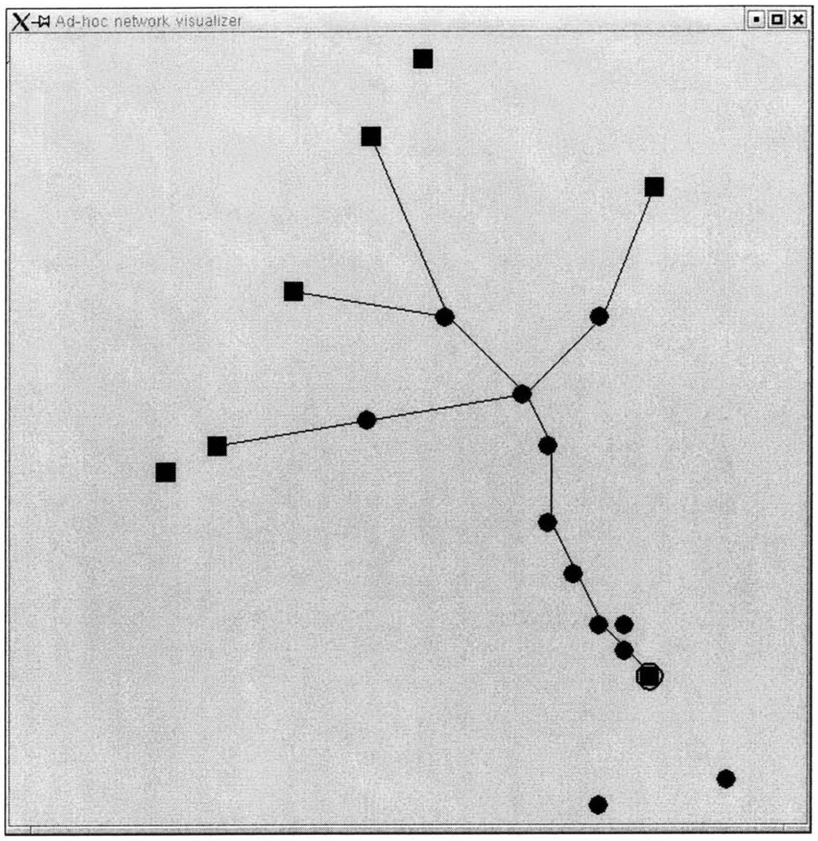


Figure 6-2: Using the directional routing policy, packets often become congested on the trunk of the network tree. Sources are shown as squares, mobile nodes are circles, and the receiver is an encircled square.

This is due to the fact that Q-routing will create alternate paths to the receiver as soon as a path becomes congested. Thus, packets will be less likely to be dropped due to buffer overflow caused by path congestion or limited buffer sizes since alternate paths will be constructed. In directional routing, on the other hand, often certain paths will become overloaded with traffic, causing significant packet drop.

Q-routing outperforms directional routing even in cases where buffer size is not a direct constraint, such as the case shown in Figure 6-1 where the buffer size is 10. This illustrates the underlying problem of network capacity. Since the total source packet generation rate exceeds the transmission rate of any one inter-node link, directional routing may still run into trouble with bottleneck links regardless of buffer size. The network capacity achieved using Q-routing is larger since more

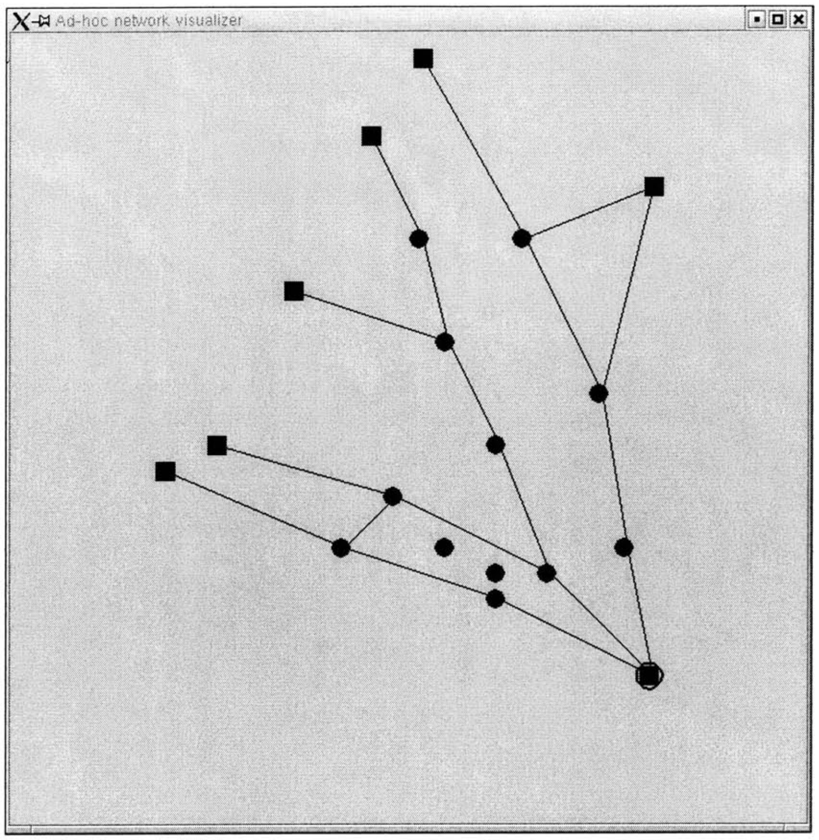


Figure 6-3: In contrast to the situation in Figure 6-2, when we use Q-routing on the same experimental setup (note that the source and receiver nodes are in the same position as the both figures), the mobile nodes in the ad-hoc network spread out to distribute packet load. Both figures show the simulator after 10000 periods, using the same initialization and movement files.

alternate paths are created around such bottlenecks. Furthermore, directional routing is unable to find circuitous paths from sources to the receiver. Since it only routes packets to neighbors that are in the direction of the receiver, any path that requires a packet to be forwarded away from the receiver for one or more hops will never be found.

These comparisons are done using a fixed movement policy we will call the *centroidal movement* policy. Under this policy, a node that is holding a connection will attempt to move to the centroid of its connected neighbors, which increases the likelihood of preserving these connections over time. If it is not holding a connection, then it simply moves about randomly searching for a new connection. Thus, the next hops determined by the routing policy strongly influence the direction of movement, since the next hops determine the node's connections to its neighbors.

When a random movement policy is used instead of the centroidal policy, Q-routing exhibits inferior performance relative to directional routing. One example is given in Figure 6-4, which shows the evolution of average system performance over time in a typical scenario. The table below gives averages over 100 different scenarios:

Movement policy	Routing policy	Average performance
Centroidal	Q-routing	.924
Centroidal	Directional	.896
Random	Q-routing	.498
Random	Directional	.519

This phenomenon is due to the fact that Q-routing influences the centroidal movement policy in a positive manner, whereas it is unable to influence a random movement policy. In some sense, Q-routing with centroidal movement is able to find circuitous source-destination paths and rope them in using the centroidal movement.

Due to this type of influence, Q-routing and directional routing result in very different configurations for the network when coupled with centroidal movement. Directional routing tends to form a network backbone, which usually comprises the most direct route to the receiver for a large portion of the source nodes. Other sources send packets towards this backbone, resulting in a tree-like network configuration, as shown in Figure 6-3. Q-routing, on the other hand, always seeks the shortest path towards the receiver, even when buffer sizes are not a constraint. This results in a fan-shaped network configuration, also shown in Figure 6-3, where each source has its own shortest path to the receiver as long as there are a sufficient number of mobile nodes to create these paths. From this observation, we can begin to see that there is an interplay between the choice of routing protocol and the movement policy of the mobile nodes.

This leads to a subtle but telling explanation for the improved performance of Q-routing over directional routing. In Q-routing, the mobile agent nodes tend to become

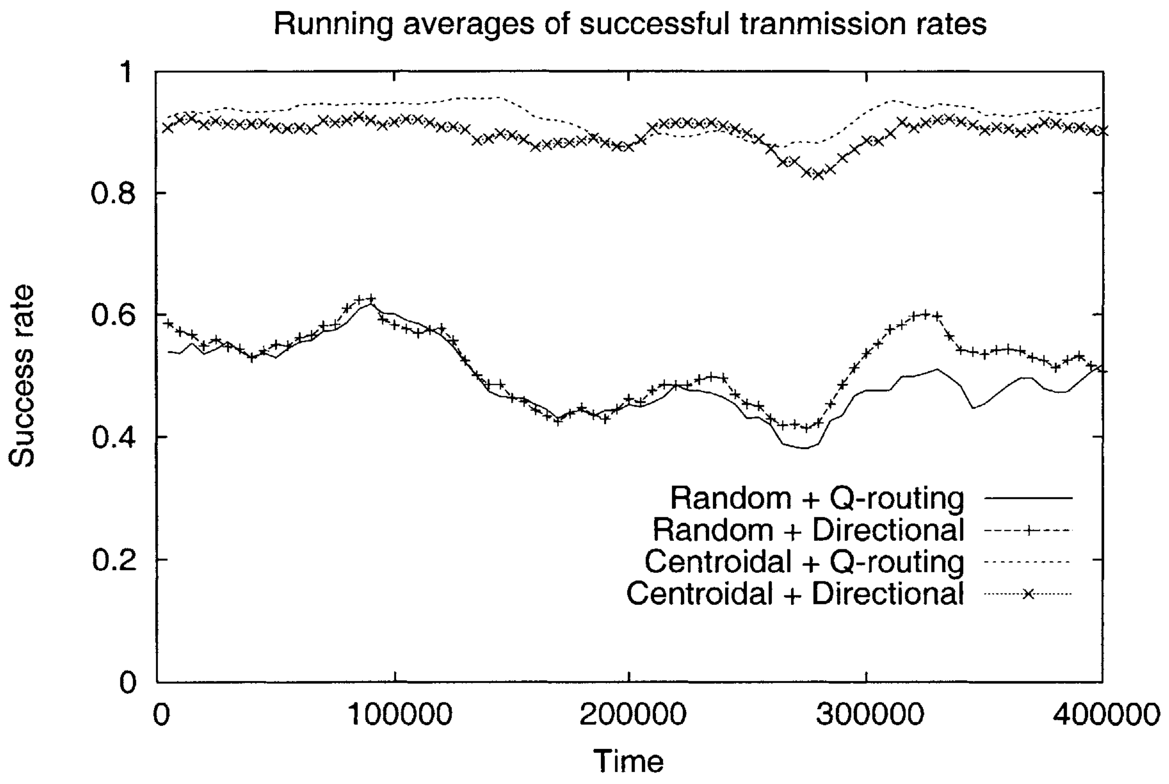


Figure 6-4: This graph shows a running average of successful transmission rates for a sample network scenario under four cases: Q-routing with centroidal movement, directional routing with centroidal movement, directional routing with random movement, and Q-routing with random movement.

more dispersed, since no network backbone is created. Thus, as source nodes move about, the Q-routing ad-hoc network is more likely to be able to remain connected without drastic reconfigurations. This interplay between routing and movement forces us to carefully consider the movement policy we choose to pair with our selected routing policy.

6.5.2 Empirical results for movement

We evaluate the performance of our learning algorithm against the centroidal movement policy given in Section 6.5.1, a hand-coded policy that uses the same observation space as the learning algorithm, and a global-knowledge central controller. Under simulation, we give the central controller access to all the node, source, and receiver positions, which would usually be unavailable to the agent nodes. Since our learning algorithm only has access to local knowledge, the central controller's performance should approximate an upper bound for the learning algorithm's performance. Moreover, this performance bound may fluctuate over time as network conditions change depending on source movement scenarios.

The central controller is designed to approximate an optimal movement policy given global knowledge about network conditions. It begins by identifying connected components among the stationary nodes. If a packet is received by any node of a connected component, then all nodes of that component can also receive the packet. However, if a packet's destination is in a different component, then the controller finds the shortest path to the destination component and recruits mobile nodes to help construct the necessary links needed to deliver the packet.

Figure 6-5 gives the average performance of a sample network scenario over time using each of these movement policies. As we can see, the learning algorithm does eventually learn a policy that behaves fairly well, but it never achieves the performance of the global knowledge controller. This is expected since the learner does not have access to the global network topology. On average, the learned policies perform slightly better than the hand-coded policy over large sets of network scenarios. However, in certain scenarios, it never learns to perform as well, possibly due to aliasing

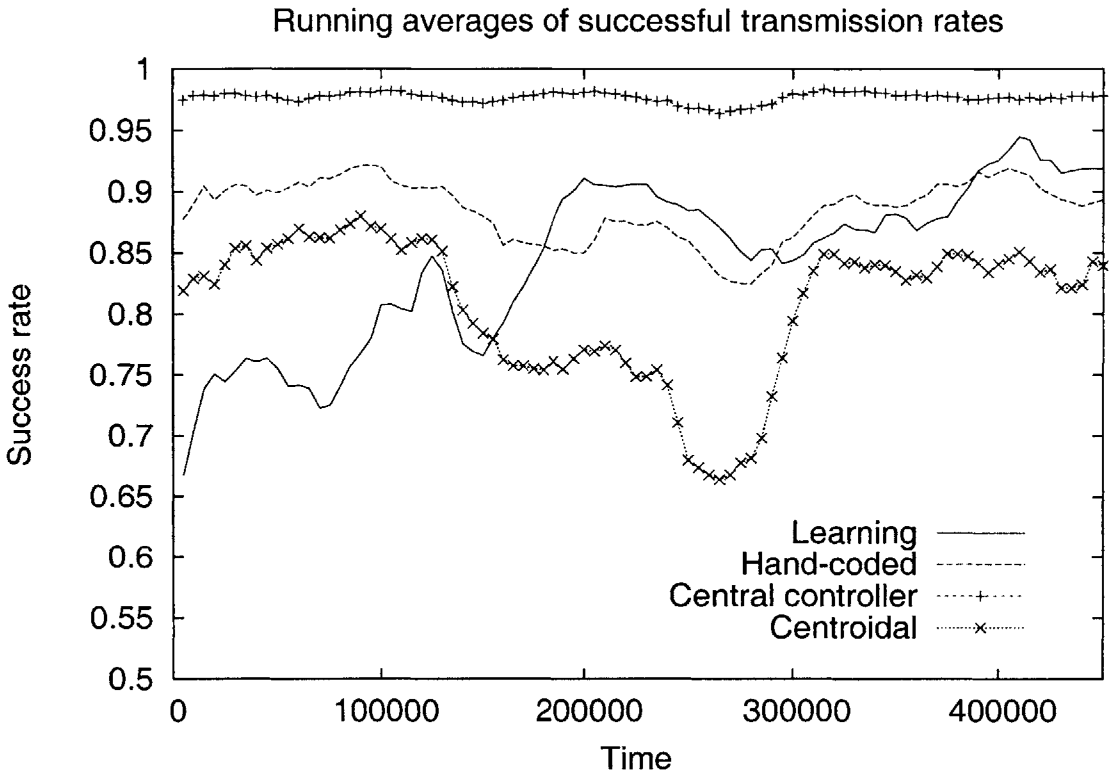


Figure 6-5: Graph showing the average performance of various movement policies over time in a typical scenario. The learning policy is shown during its training phase. The learning policy eventually exceeds the performance of the hand-coded policy that uses the same observation space, but never outperforms the global knowledge central controller.

problems.

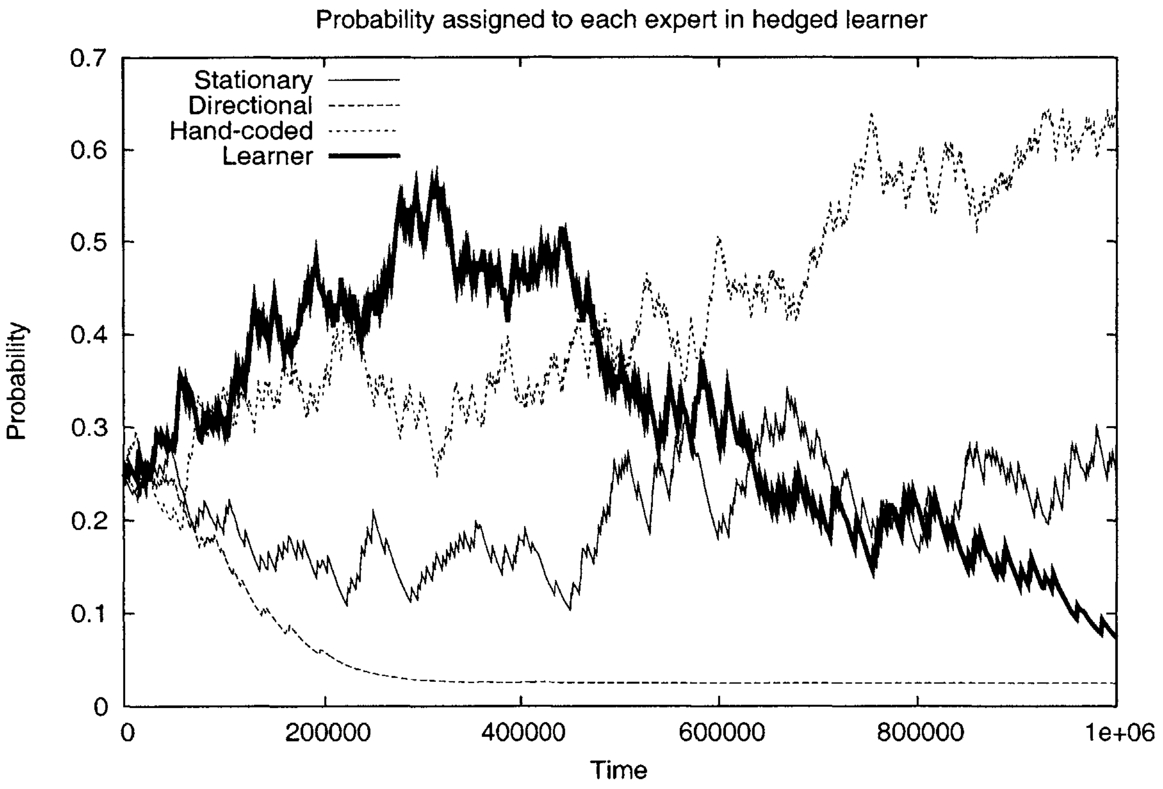


Figure 6-6: Graph showing the hedged learner's probabilities of using its various internal strategies in the mobilized ad-hoc networking domain over time. In this scenario, the pre-trained policy worked very well, and we can see that it is followed most of the time.

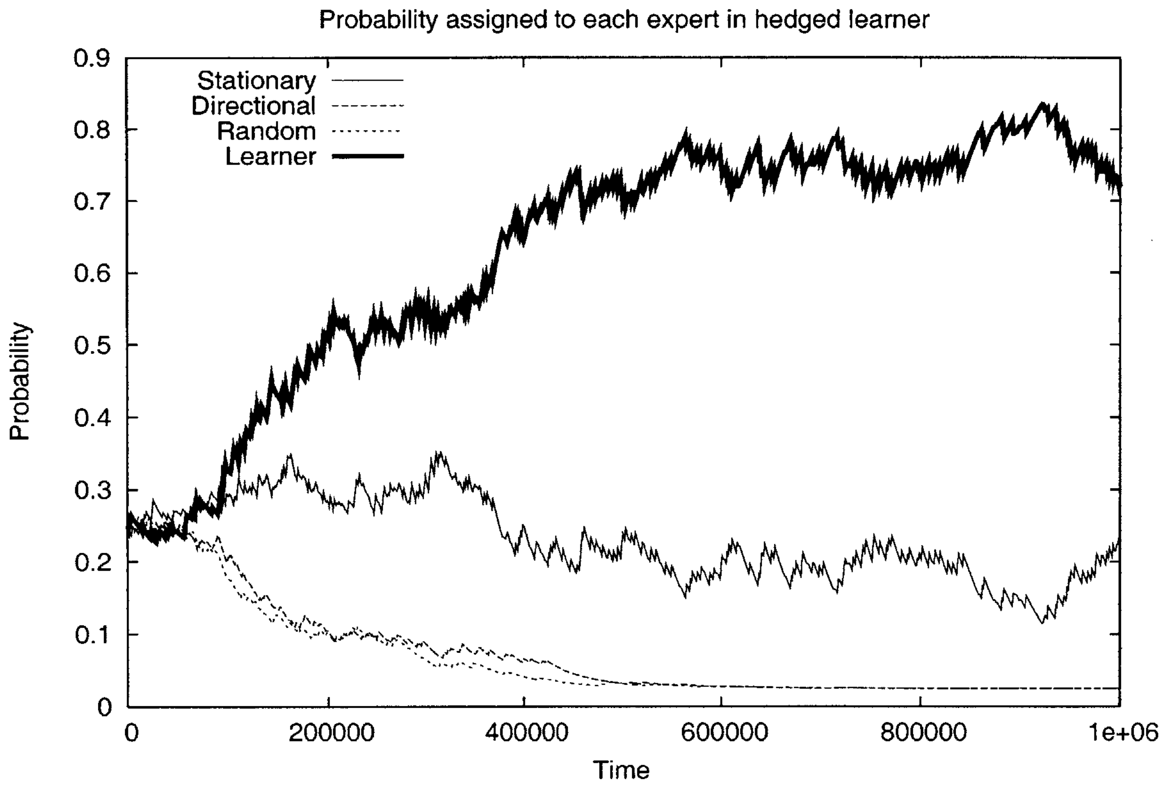


Figure 6-7: Graph showing the hedged learner's probabilities of using its various internal strategies over time. In this case, the learning algorithm succeeds in learning a better policy than any of the fixed strategies.

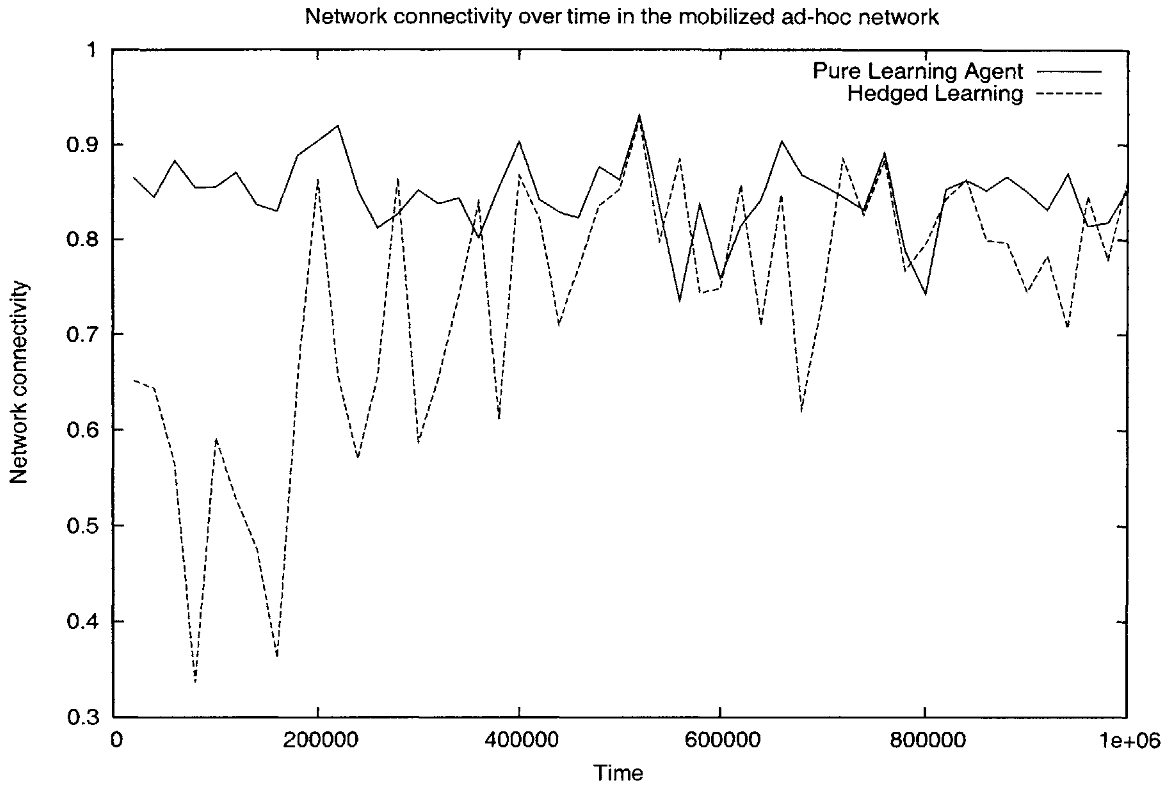


Figure 6-8: Relative to the performance of the system when the nodes are purely using the learning algorithm, the hedged learner suffers some loss since it must spend time evaluating its other possible policies. It also continues to suffer higher variance in its performance even once it has assigned high probability to following the learning algorithm, since it still evaluates the other policies occasionally.

Chapter 7

Conclusions and future work

The thesis seeks to develop a fundamental understanding of multi-agent interactions, and to build upon that understanding to design large, complex systems. Our initial investigations focused on simple two-player repeated matrix games, where we can concentrate on analyzing player interactions rather than environmental interactions. There had been much confusion about the goals that learning agents ought to try to achieve in multi-agent settings, and equilibrium concepts were popular. In contrast, we noted that equilibrium concepts are often ill-suited for competitive games in practice, since there is no reason to expect that opponents would possess perfect rationality, nor would the opponents presume that our agents were perfectly rational either. Attempting to compute a best response strategy against a general, unrestricted opponent can also be shown to be futile. We demonstrated that an agent's beliefs about its potential types of opponents drives the design of good algorithms, and showed that even state-of-the-art algorithms could be exploited due to their opponent assumptions.

As the environment grows more complex, and we move beyond simple two-player repeated games, we need to bring in more sophisticated techniques for designing practical, realistic, multi-agent systems. We show that filtering methods from control theory, equilibrium concepts from game theory, universal prediction from information theory, experts algorithms from online learning, and reinforcement learning methods such as policy gradient estimation and Q-learning can be combined to create effective learning algorithms for optimizing multi-agent behaviors.

Different combinations of methods are suited for various special situations. For example, in cooperative environments where we are trying to train agents using a global reward signal, we may be able to abstract away much of the complexity of the multi-agent domain. Instead of explicitly modeling the effect of the other agents, we assume their combined effect on the global reward signal is encapsulated by a single noise term. We introduced the concept of reward filtering, in which methods such as Kalman filtering can be used to estimate and remove unwanted noise from the reward signal, thus allowing each agent to learn a good policy.

In non-cooperative settings, we introduced the idea of hedged learning, and proposed algorithms that seek to combine the efficiency of learning with opponents models, and the performance guarantees obtained from hedging, or regret-minimizing, methods. These methods are useful in situations where we know we may face large classes of potential opponents, and we have multiple different models to describe their behavior. Hedged learning allows us to hedge over these possible models, while guaranteeing that even if all the models are incorrect, and we are in fact facing a novel, highly intelligent opponent, we will still perform reasonably well. We gave performance guarantees and demonstrated this algorithm's effectiveness in several sample domains.

One of the main application areas we focused on is mobilized ad-hoc networking, and we've shown that these techniques can be applied to train mobile agent nodes to form an effective ad-hoc network that can track moving sources and route information from these sources to certain receiver nodes. A

A number of specific future goals stem from this work. First, we plan to continue to apply the theory and techniques we've developed to the design of mobilized ad-hoc networks, and to derive new insights and potentially new algorithms from studying problems in this domain. Up to now, we have treated this domain as a fully cooperative scenario in which mobile nodes collaborate to form an ad-hoc network. However, insights learned from our work on competitive multi-agent settings may be applied here to consider networks in non-cooperative environments, such as in the presence of jamming signals or within a market-based bandwidth allocation process.

Secondly, our algorithms seem to hold great promise for other application areas such as financial equity markets or commodity auctions. In particular, the hedged learning approach provides an explicit way to combine expert knowledge with regret-minimizing behavior, possibly yielding techniques that can provide good performance guarantees while empirically producing even better results in real applications. Whereas traditional online resource allocation algorithms tend to base action choices completely on hindsight and past performance, these hybrid algorithms incorporate the predictive power of learning experts in order to assess the future ramifications of current choices based on learned models of the environment.

Thirdly, given any good strategy, we know that it can be exploited if known by the opponent in a competitive setting. Thus, obscuring or obfuscating our strategy may be an important additional consideration in agent design. Obfuscation may take the form of simple randomization over possible strategies, or we might employ deception techniques such as those we demonstrated can be used to exploit the weaknesses of certain state-of-the-art repeated-game players. We would like to quantify the value of obfuscating our strategies and devise techniques that provide a clear tradeoff or synergy between obfuscation and performance. The question of exploration versus exploitation has always been important to reinforcement learning; perhaps in multi-agent situations, another axis, obfuscation, needs to also be considered.

More broadly, my long-term research will attempt to bridge the crucial gap between game theoretic equilibria and practical methods for achieving good performance with or without convergence. My research into categorizing and intelligently combining current multi-agent learning methods is a step in this direction. We need to further understand the differing levels of opponent modeling that are necessary for agents to succeed in different domains.

7.1 Richer regret

In Chapters 3 and 5, we established the usefulness of using regret as a criterion for measuring the performance of multi-agent learning algorithms. We proposed the technique of hedged learning, where we hedge between behaviors based on various opponent models. We show upper bounds on the regret when this algorithm is used, and we further show that the algorithm's actual performance in experimental trials is more robust than other previous techniques against a wide range of opponents.

However, this work still does not exploit the full potential of hedged learning. We have only explored the use of relatively simple opponent models, and we have not made use of the extra information that might be conveyed when following one expert's recommendations. For example, if we could use richer opponent models that allow us to infer probable outcomes of strategies that were not actually tested, this would increase the efficiency of our algorithm considerably. Both the upper bounds on regret and the actual performance of the algorithm should be improved in this case.

7.2 The swarm

The ultimate validation of a learning system is to evaluate its performance on real, live robots. The *swarm* is a group of up to 100 miniature mobile robots, each capable of operating autonomously and communicating with other robots in the swarm. Using these capabilities, we can program the robots to collaborate and produce interesting behaviors. For example, simple tasks such as "follow the leader" or "orbit around the leader" are easy to implement using the swarm programming library. However, even programming these simple tasks often involves hand-tuning parameters that affect the system performance greatly. The task of custom-tuning a more complicated program quickly becomes cumbersome. Thus we might wish to employ learning algorithms to train the swarm instead. Using simple, easy-to-program functions as the basic building blocks of swarm behavior, we can train the swarm to collaborate and achieve complex goals without the need to engage in tiresome hand-tuning and custom-coding for each new behavior we wish to produce.

This type of learning can be beneficial in a wide variety of situations. For example, we may wish to deploy the swarm into different types of environments that may require different behaviors for the swarm to achieve its goals. Rather than hand-crafting behaviors for the swarm to follow in each of the environments, it may be easier to simulate the environments and allow the swarm to learn good behaviors on its own in simulation.

The swarm robots are designed to allow us to run hardware simulations rather than relying simply on software. Each swarm robot is fitted with IR communications, bump skirts, and a small video camera. They are capable of operating for 2-3 hours on a single charge, and are able to autonomously recharge at a docking station. Developed by iRobots, the Swarm is intended to be a hardware simulation platform that ideally needs very little intervention from its human operators. The Swarm toolbox consists of a library of functions that implement simple swarm behaviors such as follow-the-leader, follow-the-gradient, orbit-a-robot, or disperse. These can be used as the building blocks for designing more complex behaviors. For example, in one scenario, the swarm robots were used explore inside a large building by forming a connected web throughout the interior spaces. [McLurkin and Smith, 2004]

7.2.1 Swarm challenges

Part of the challenge (and the excitement) of working with real robots is that we can no longer conform our simulation to fit our models of the world. We must now conform our models and our learning algorithms to the real world situation that we are given, as embodied by the robots and the environment they find themselves in. Our challenges arise from the various constraints that we then discover, and robots have no shortage of constraints.

First of all, unlike in simulation, we can no longer afford to spend hundreds of thousands of iterations in training. Algorithms must perform well in a reasonable span of time. The swarm robots of capable of approximately four rounds of communication each second. On a single charge, they can operate for two to three hours, and while they are capable of autonomously recharging, in practice we require a human to watch over the robots for the duration of any experiment. That puts a practical limit of eight or so hours for an experiment, which translates to 115,200 iterations at best.

Constraint 1. The swarm must be able to learn reasonable behavior within 100,000 iterations.

Secondly, communication failures and robot breakdowns are more common than we would like to assume in a perfect simulated world. Packet losses are almost 25%, although luckily some low-level routines are able to abstract away most of that lossiness. At the level of the learning algorithms, we can expect a packet loss rate of about 5%.

Robots also simply stop working, due to equipment malfunction, some unusual obstacles in the environment, or a variety of other reasons.

Constraint 2. Communications are lossy, and robots break down.

Onboard memory and processing power is provided by (fill in blank). Thus, our algorithms are limited in their complexity, both in terms of running time and memory allocation.

Constraint 3. Processing power is limited.

As discussed earlier, the swarm robots are only able to emit two to four information-bearing packets each second. Each packet can contain several bytes of information, so the number of actual variables transmitted between swarm robots is not the main constraint. Rather, the constraint is due to the fact that these variables can only be updated between robots a few times each second. Luckily, the robots also do not move very fast (fill in here exactly how fast), so state variables usually do not change very quickly.

Constraint 4. Bandwidth is limited.

7.3 Conclusion

This thesis fuses ideas from machine learning, game theory, distributed systems, and networking to explore the relatively new field of multi-agent learning. From a seemingly simple problem involving the competitive interaction between two intelligent agents to the complex design of large multi-agent systems, ideas from each of these fields can provide important insights. In the coming years, as computational power proliferates and embeds itself in all aspects of our lives, the need to understand multi-agent interactions and multi-agent systems will become increasingly important. Learning algorithms provide us with valuable tools for studying these systems and designing better systems. Their wide applicability ranges from web-based agent systems to self-interested agents in financial markets to the behavior of teams of robots.

Bibliography

[Aberdeen, 2002] D. Aberdeen. Policy-gradient algorithms for partially observable markov decision processes. *Ph.D. Thesis, Australian National University*, 2002.

[Auer *et al.*, 1995] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995.

[Auer *et al.*, 2002] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. The non-stochastic multi-armed bandit problem, 2002.

[Baxter and Bartlett, 2001] Jonathan Baxter and Peter Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381, 2001.

[Bernstein *et al.*, 2002] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002.

[Bertsekas and Tsitsiklis, 1996] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic Programming.* Athena Scientific, 1996.

[Blum and Kalai, 1997] Blum and Kalai. Universal portfolios with and without transaction costs. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 1997.

[Blum *et al.*, 2003] B. Blum, C. R. Shelton, and D. Koller. A continuation method for nash equilibria in structured games. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[Bowling and Veloso, 2002a] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[Bowling and Veloso, 2002b] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[Boyan and Littman, 1994] J. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in NIPS*, 1994.

[Cesa-Bianchi and Lugosi, 2003] N. Cesa-Bianchi and G. Lugosi. Potential-based algorithms in on-line prediction and game theory. *Machine Learning*, 51(3):239–261, 2003.

[Chang and Kaelbling, 2002] Y. Chang and L. P. Kaelbling. Playing is believing: The role of beliefs in multi-agent learning. In *Advances in Neural Information Processing Systems*, 2002.

[Chang and Kaelbling, 2005] Y. Chang and L. P. Kaelbling. Hedged learning: Regret minimization using learning experts. In *International Conference on Machine Learning, submitted*, 2005.

[Chang *et al.*, 2003] Y. Chang, T. Ho, and L. P. Kaelbling. Reinforcement learning in mobilized ad-hoc networks. *Technical Report, AI Lab, MIT*, 2003.

[Chang *et al.*, 2004a] Y. Chang, T. Ho, and L. P. Kaelbling. All learning is local: Multi-agent learning in global reward games. In *Advances in Neural Information Processing Systems*, 2004.

[Chang *et al.*, 2004b] Y. Chang, T. Ho, and L. P. Kaelbling. A reinforcement learning approach to mobilized ad-hoc networks. *International Conference on Autonomic Computing*, 2004.

[Chatzigiannakis *et al.*, 2001] I. Chatzigiannakis, S. Nikoletseas, N. Paspallis, P. Spirakis, and C. Zaroliagis. An experimental study of basic communication protocols in ad-hoc mobile networks. In *5th Workshop on Algorithmic Engineering*, 2001.

[Chen *et al.*, 2001] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance. In *SIGMOBILE*, 2001.

[Choi *et al.*, 1999] S. Choi, D. Yeung, and N. Zhang. Hidden-mode Markov decision processes. In *IJCAI Workshop on Neural, Symbolic, and Reinforcement Methods for Sequence Learning*, 1999.

[Claus and Boutilier, 1998] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiaent systems. In *Proceedings of the 15th AAAI*, 1998.

[de Farias, 2004] D. P. de Farias. How to combine expert (or novice) advice when actions impact the environment. In *Proceedings of NIPS*, 2004.

[Freund and Schapire, 1995] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[Freund and Schapire, 1999] Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[Fudenburg and Levine, 1995] Drew Fudenburg and David K. Levine. Consistency and cautious fictitious play. *Journal of Economic Dynamics and Control*, 19:1065–1089, 1995.

[Greenwald and Hall, 2003] A. Greenwald and K. Hall. Correlated q-learning. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

[Grossglauser and Tse, 2001] M. Grossglauser and D. Tse. Mobility increases the capacity of ad-hoc wireless networks. In *INFOCOM*, 2001.

[Gupta and Kumar, 1998] P. Gupta and P. R. Kumar. Capacity of wireless networks. In *Stochastic Analysis, Control, Optimization, and Applications*. Birkhauser, 1998.

[Hall and Greenwald, 2001] Keith Hall and Amy Greenwald. Correlated q-learning. In *DIMACS Workshop on Computational Issues in Game Theory and Mechanism Design*, 2001.

[Hart and Mas-Colell, 2001] S. Hart and A. Mas-Colell. A general class of adaptive strategies. *Journal of Economic Theory*, 98(1):26–54, 2001.

[Hsu *et al.*, 2005] D. Hsu, G. Snchez-Ante, and Z. Sun. Hybrid prm sampling with a cost-sensitive adaptive strategy. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.

[Hu and Wellman, 1998] Junling Hu and Michael P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the 15th Int. Conf. on Machine Learning (ICML-98)*, 1998.

[Johnson and Maltz, 1996] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, volume 353. 1996.

[Kaelbling *et al.*, 1996] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Kakade and Ng, 2004] S. Kakade and A. Ng. Online bounds for bayesian algorithms, 2004.

[Kalman, 1960] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the American Society of Mechanical Engineers, Journal of Basic Engineering*, 1960.

[Kearns and Singh, 1998] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Proceedings of the 15th International Conference on Machine Learning*, pages 260–268, 1998.

[Li and Rus, 2000] Q. Li and D. Rus. Sending messages to mobile users in disconnected ad-hoc networks. In *MOBICOM*, 2000.

[Littlestone and Warmuth, 1989] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *IEEE Symposium on Foundations of Computer Science*, pages 256–261, 1989.

[Littman and Stone, 2001] Michael Littman and Peter Stone. Leading best-response stratgies in repeated games. In *17th Int. Joint Conf. on Artificial Intelligence (IJCAI-2001) workshop on Economic Agents, Models, and Mechanisms*, 2001.

[Littman and Szepesvári, 1996] Michael L. Littman and Csaba Szepesvári. A generalized reinforcement-learning model: Convergence and applications. In *Proc. of the 13th ICML*, 1996.

[Littman *et al.*, 2001] M. Littman, M. Kearns, and S. Singh. An efficient exact algorithm for singly connected graphical games. In *Proceedings of NIPS*, 2001.

[Littman, 1994] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. of the 11th ICML*, 1994.

[Littman, 2001] Michael L. Littman. Friend-or-foe q-learning in general-sum games. In *Proceedings of the 18th Int. Conf. on Machine Learning (ICML-01)*, 2001.

[Loch and Singh, 1998] John Loch and Satinder Singh. Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *Proc. 15th International Conf. on Machine Learning*, pages 323–331. Morgan Kaufmann, San Francisco, CA, 1998.

[Mannor and Shimkin, 2001] Shie Mannor and Nahum Shimkin. Adaptive strategies and regret minimization in arbitrarily varying Markov environments. In *Proc. of 14th COLT*, 2001.

[Mannor and Shimkin, 2003] S. Mannor and N. Shimkin. The empirical bayes envelope and regret minimization in stochastic games. *Mathematical Operation Research*, 28(2):327–345, 2003.

[McLurkin and Smith, 2004] J. McLurkin and J. Smith. Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In *Proceedings of DARS*, 2004.

[Merhav and Feder, 1998] Merhav and Feder. Universal prediction. *IEEETIT: IEEE Transactions on Information Theory*, 44, 1998.

[Monteleoni and Jaakkola, 2003] C. Monteleoni and T. Jaakkola. Online learning of non-stationary sequences. 2003.

[Nachbar and Zame, 1996] J.H. Nachbar and W.R. Zame. Non-computable strategies and discounted repeated games. *Economic Theory*, 1996.

[Nagayuki *et al.*, 2000] Yasuo Nagayuki, Shin Ishii, and Kenji Doya. Multi-agent reinforcement learning: An approach based on the other agent's internal model. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-00)*, 2000.

[Ortiz and Kearns, 2002] L. Ortiz and M. Kearns. Nash propagation for loopy graphical games. In *Proceedings of NIPS*, 2002.

[Papadimitriou, 2001] C. H. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of STOC*, 2001.

[Perkins and Royer, 1997] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *MILCOM Panel*, 1997.

[Peshkin *et al.*, 2000] L. Peshkin, K. E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence*, 2000.

[Peshkin, 2002] L. Peshkin. Reinforcement learning by policy search. *Ph.D. Thesis, MIT*, 2002.

[Schmidhuber, 1999] J. Schmidhuber. A general method for incremental self-improvement and multi-agent learning. *Evolutionary Computation: Theory and Applications*, pages 81–123, 1999.

[Singh *et al.*, 2000] S. Singh, M. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, 2000.

[Stone and Sutton, 2001] P. Stone and R. S. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *ICML*, 2001.

[Sutton and Barto, 1999] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1999.

[Szita *et al.*, 2002] Istvan Szita, Balimt Takacs, and Andras Lorincz. e-mdps: Learning in varying environments. *Journal of Machine Learning Research*, 2002.

[Vovk, 1998] V. Vovk. A game of prediction with expert advice. *Journal of Computer and System Sciences*, 56:153–173, 1998.

[Watkins, 1989] C. J. Watkins. Learning with delayed rewards. *Ph.D. Thesis, University of Cambridge*, 1989.

[Williams, 1992] R. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.