Reinforcement Learning Without Rewards

Umar Ali Syed

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Robert E. Schapire

September 2010

Abstract

Machine learning can be broadly defined as the study and design of algorithms that improve with experience. *Reinforcement learning* is a variety of machine learning that makes minimal assumptions about the information available for learning, and, in a sense, defines the problem of learning in the broadest possible terms. Reinforcement learning algorithms are usually applied to "interactive" problems, such as learning to drive a car, operate a robotic arm, or play a game. In reinforcement learning, an autonomous agent must learn how to behave in an unknown, uncertain, and possibly hostile environment, using only the sensory feedback that it receives from the environment. As the agent moves from one state of the environment to another, it receives only a *reward* signal — there is no human "in the loop" to tell the algorithm exactly what to do. The goal in reinforcement learning is to learn an optimal behavior that maximizes the total reward that the agent collects.

Despite its generality, the reinforcement learning framework does make one strong assumption: that the reward signal can always be directly and unambiguously observed. In other words, the feedback a reinforcement learning algorithm receives is assumed to be a part of the environment in which the agent is operating, and is included in the agent's experience of that environment. However, in practice, rewards are usually manually-specified by the practitioner applying the learning algorithm, and specifying a reward function that elicits the desired behavior from the agent can be a subtle and frustrating design problem. *Our main focus in this thesis is the design and analysis of reinforcement learning algorithms which do* not *require complete knowledge of the rewards.* The contributions of this thesis can be divided into three main parts:

- In Chapters 2 and 3, we review the theory of *two-player zero-sum games*, and present a novel analysis of existing *no-regret* algorithms for solving these games. Our results show that no-regret algorithms can be used to compute strategies

in games that satisfy a much stronger definition of optimality than is commonly used.

- In Chapters 4 and 5, we present new algorithms for *apprenticeship learning*, a generalization of reinforcement learning where the true rewards are unknown. The algorithms described in Chapter 5 will leverage the game-theoretic results from Chapters 2 and 3.

- In Chapter 6, we show how partial knowledge of the rewards can be used to accelerate *imitation learning*, an alternative to reinforcement learning where the goal is to imitate another agent in the environment.

In summary, we design and analyse several new algorithms for reinforcement learning that do not require access to a fully observable or fully accurate reward signal, and by doing so, add considerable flexibility to the traditional reinforcement learning framework.

Acknowledgements

Rob Schapire's positive qualities as an advisor are too numerous to briefly summarize, but I will try. He is always calm, cool, and relaxed. When a problem seems insoluable, he does not worry. When we are facing a paper deadline, he does not panic. He never said anything negative about my research that was not also constructive and tactfully phrased. He has an ability to quickly process my often rambling and ill-formed ideas, and then immediately offer a helpful suggestion. Someone else once described Rob's talent best: He does not say much, but he never, ever says anything wrong. I was extremely lucky to have had Rob Schapire as a mentor, teacher and friend, and my only regret is that I am unlikely to have a collegue of his caliber again.

It was great fun to work along side my fellow students Jordan Boyd-Graber, Berk Kapicioglu, Jonathan Chang, Indraneel Mukherjee, Miro Dudik, and Melissa Carroll. I hope we stay in touch, both personally and professionally. I am particularly grateful to Ronny Luss for suggesting that I study lexicographic optimality, a topic which eventually grew into a large part of this thesis.

Although our research together does not appear in this thesis, I loved working with Jen Rexford, Alex Fabrikant, Howard Karloff and Gordon Wilfong on BGP puzzles. I especially benefited from the mentorship of Jen Rexford, who had endless tolerance for my divided and distracted attention.

I was fortunate to have had two rewarding summer internships while at Princeton. Jason Williams, my mentor at AT&T, was an excellent model for me of a successful young researcher. He is driven and focused, but with a sunny and friendly disposition, and I hope to emulate his career. Srini Bangalore, Patrick Haffner and Mazin Gilbert also gave me terrific advice. At Microsoft, I was lucky to work closely with Nina Mishra, Aleks Slivkins, and Alan Halverson, who are as passionate about research as they are talented. I learned much more from them than they probably realize.

I am grateful to the members of my committee, Dave Blei, Warren Powell, Michael

Littman and Yael Niv, for their suggestions on improving this thesis, which I have tried to heed closely.

My research would not have been possible without the generous funding of the National Science Foundation (under grant IIS-0325500), and a Wallace Memorial Fellowship in Engineering.

Abu, Ammi and Safia can attest that my only aspiration as a young man was to sleep in as late as possible. They helped me aim a little higher, and my long and circuitous journey to a Ph.D. would not have been possible without their love and support. I guess one never repay the debt that is owed to one's parents, but this thesis is my small attempt to do so.

As challenging as it is sometimes to be a Ph.D. student, it is even more difficult to be married one. It certainly requires more patience. Sana suffered so many of my long, late and uncertain nights at the office that she must have felt as though she was married to a ghost, but she responded only with love, grace and good humor. I am blessed to married to her. As small token of my appreciation, I will also acknowledge our two cats, Cosmo and Bonsai.

Say, "I do not ask you for any reward. All I seek is to help you find the right path to your Lord, if that is what you choose."

Quran 25:57

Contents

Chapter 1

Overview

Machine learning can be broadly defined as the study and design of algorithms that improve with experience. Machine learning algorithms have been enormously successful in domains as varied as data mining, natural language understanding, and molecular biology.

"Machine learning" is really an umbrella term that covers several approaches to designing learning algorithms, each with its own strengths and weaknesses. To illustrate these differences, consider a specific problem which machine learning algorithms are often used to solve: filtering email spam. Different kinds of learning algorithms make different assumptions about what information is available for spam filtering. In *supervised learning*, one assumes that a human is willing to supply a data set containing many emails, each manually labeled "spam" or "non-spam". In *semi-supervised* learning, only some of the emails in the data set need to be labeled — a useful advantage because the cost of labeling an example is typically expensive. Similarly, in *active learning*, a very small number of emails are labeled in an incremental fashion, and only at the request of the learning algorithm.

Reinforcement learning is an approach to learning that makes minimal assumptions about the information available for learning, and, in a sense, defines the problem

of learning in the broadest possible terms. Reinforcement learning algorithms are not usually applied to "static" problems, like predicting the content of an email, but more often to "interactive" problems, such as learning to drive a car, operate a robotic arm, or play a game. In reinforcement learning, an autonomous agent must learn how to behave in an unknown, uncertain, and possibly hostile environment, using only the sensory feedback that it receives from the environment. As the agent moves from one state of the environment to another, it receives only a *reward* signal — there is no human "in the loop" to tell the algorithm exactly what to do. The goal in reinforcement learning is to learn an optimal behavior that maximizes the total reward that the agent collects.

As an example, consider the task of learning to navigate a car through an obstacle course. A typical reinforcement learning algorithm will begin this task with little knowledge of the environment or the ultimate goal. As the algorithm executes, the car is driven by the algorithm, randomly at first, but then more purposefully over time, while receiving rewards for its actions. It may, for instance, receive negative reward for hitting obstacles and positive reward for crossing the finish line. By using this feedback, after many trials the algorithm will have learned a driving behavior that leads the car to the finish line while avoiding as many obstacles as possible. This learning process closely resembles — and was in fact motivated by — the way humans and animals learn. Indeed, the reinforcement learning framework is often used as a descriptive model in psychology and neuroscience.

Despite its generality, the reinforcement learning framework does make one strong assumption: that the reward signal can always be directly and unambiguously observed. In other words, the feedback a reinforcement learning algorithm receives is assumed to be a part of the environment in which the agent is operating, and is included in the agent's experience of that environment. However, in practice, rewards are usually manually-specified by the practitioner applying the learning algorithm,

and specifying a reward function that elicits the desired behavior from the agent can be a subtle and frustrating design problem. *Our main focus in this thesis is the design and analysis of reinforcement learning algorithms which do* not *require complete knowledge of the rewards.* The contributions of this thesis can be divided into three main parts:

- In Chapters 2 and 3, we review the theory of *two-player zero-sum games*, and present a novel analysis of existing *no-regret* algorithms for solving these games. Our results show that no-regret algorithms can be used to compute strategies in games that satisfy a much stronger definition of optimality than is commonly used.

- In Chapters 4 and 5, we present new algorithms for *apprenticeship learning*, a generalization of reinforcement learning where the true rewards are unknown. The algorithms described in Chapter 5 will leverage the game-theoretic results from Chapters 2 and 3.

- In Chapter 6, we show how partial knowledge of the rewards can be used to accelerate *imitation learning*, an alternative to reinforcement learning where the goal is to imitate another agent in the environment.

We now review our contributions in greater detail.

Game theory is the study of the behavior of agents who strategically interact to achieve their goals. Game theory has had a huge impact on many fields, such as economics, psychology, and evolutionary biology. It has been particularly influential in machine learning, because many machine learning algorithms can be interpreted from a game-theoretic perspective. At the same time, many problems in game theory can be efficiently solved by techniques that were originally designed for machine learning problems. Chapter 2 contains a review of the key ideas and results from the theory of two-player zero-sum games, a type of game in which the goals of two agents are

diametrically opposed. We also describe several algorithms for computing optimal strategies in zero-sum games.

In Chapter 3, we present a novel analysis of existing no-regret algorithms for computing optimal strategies in games. Our analysis will reveal that no-regret algorithms are particularly well-suited to computing strategies that take advantage of weaknesses in an adversary's strategy, a property known as *lexicographically optimality*. We prove that, under certain technical assumptions about the structure of the game, a well-known no-regret algorithm called the *MW algorithm* can be used to compute lexicographically optimal strategies in zero-sum games. In fact, under these assumptions, it can be used to compute an optimal strategy that is *pure*, a particularly simple type of strategy. We also describe a set of basic conditions which, if satisfied by any algorithm (MW or otherwise), allow the technical assumptions about the game to be relaxed.

Beginning in Chapter 4, we turn to the problem designing reinforcement learning algorithms that do not require complete knowledge of the rewards. Many of our contributions are made within the apprenticeship learning framework, a recently proposed generalization of reinforcement learning. In apprenticeship learning, the goal of the learning agent is to earn a large amount of reward relative to the behavior of an observed *mentor*, though the true rewards are unknown.

Existing apprenticeship learning algorithms are premised on mimicking the mentor, and in Chapter 4 we present algorithms that take this approach. We offer two main contributions. First, we present an apprenticeship learning algorithm that, like existing algorithms, assumes that the true rewards belong to certain linear family, but is considerably faster and simpler than existing algorithms. Next, we remove the linearity assumption and prove that apprenticeship learning is possible even in a situation where almost nothing is assumed about the rewards. To accomplish this, we reduce apprenticeship learning to a supervised learning problem. Not surprisingly, in

both cases, the goodness of the behavior learned by our algorithm depends strongly on the goodness of the behavior exhibited by the mentor.

In Chapter 5, instead of developing apprenticeship learning algorithms that mimic the mentor, we apply the game-theoretic results from Chapters 2 and 3. Our approach is to assume that the unknown rewards are controlled by an adversary, which allows us to use the MW algorithm to solve the apprenticeship learning problem in substantially less time than existing algorithms. We explain how our game-theoretic formulation, and especially our results about lexicographic optimality, imply that the MW algorithm can sometimes learn considerably better behavior than that demonstrated by the mentor, a property that mimicking-based apprenticeship learning algorithms do not share. In fact, the algorithm can be easily modified to learn a certain "conservative" behavior in case no mentor demonstrations are available. Moreover, our results about lexicographic optimality imply that the MW algorithm is particularly insensitive to the "units" in which the rewards are expressed. Our game-theoretic formulation also yields a straightforward algorithm, based on linear programming, that can be used to quickly solve the apprenticeship learning problem in cases where an explicit description of the environment is given.

The objective of apprenticeship learning, as in reinforcement learning, is to maximize reward, and all the algorithms we have discussed thus far are designed for this objective. But in some applications, it is more sensible to *imitate* the mentor exactly, regardless of how much reward that behavior earns. Unfortunately, even the mimicking-based apprenticeship learning algorithms described in Chapter 4 are not designed to exactly reproduce the behavior exhibited by the mentor. In Chapter 6, we develop an algorithm whose explicit goal is to imitate the mentor. What role can rewards play in such a problem setting? Our approach is to assert an *a priori* belief that the mentor is behaving in a manner that earns large reward, and use this prior knowledge to guide our imitation of the mentor's behavior. In a sense, this approach

turns the apprenticeship learning problem on its head: instead of using the mentor's behavior to learn the true rewards, we use our best guess of the true rewards to learn the mentor's behavior. We apply our method to the problem of modeling the behavior of users in a spoken dialog system, who typically behave in a goal-directed manner, and show that our algorithm accelerates the learning of their behavior.

In summary, our main contribution in this thesis is to design and analyse several new algorithms for reinforcement learning that do not require access to a fully observable or fully accurate reward signal, and by doing so, add considerable flexibility to the traditional reinforcement learning framework.

1.1 Notational Conventions

A variable written in bold lower case, such as $\mathbf{p}$, denotes a vector, while a variable written in bold upper case, such as $\mathbf{M}$, denotes a matrix. We write $p(i)$ for the ith component of the vector $\mathbf{p}$, and $M(i, j)$ for the entry in the ith row and jth column of the matrix $\mathbf{M}$. In general, bold is used to distinguish between scalar and non-scalar quantities.

To reduce notational overhead, we assume that, when taking the product of a vector and a matrix, the vector is suitably transposed, though it may not be written that way. So if $\mathbf{p}$ is an $n \times 1$ vector and $\mathbf{M}$ is an $n \times m$ matrix, then $\mathbf{pM}$ means $\mathbf{p}^T\mathbf{M}$.

We write $M(i, \mathbf{q})$ to denote the ith row of the column vector $\mathbf{Mq}$, and $M(\mathbf{p}, j)$ to denote the jth column of the row vector $\mathbf{pM}$.

The expressions $E_{x \sim D}[f(x)]$ and $E[f(x) \mid x \sim D]$ are equivalent; they both denote the expected value of the function $f(x)$ when x is drawn from the distribution D. Likewise, the expressions $\Pr_{x \sim D}(E)$ and $\Pr(E \mid x \sim D)$ both denote the probability of an event E when x is drawn from the distribution D.

Chapter 2

Two-Player Zero-Sum Games

Consider the following problems:

- A chess enthusiast is playing a game against a difficult opponent. Can she play in a manner that guarantees that she will win, no matter how well her opponent plays?
- A criminal is being interrogated by the police, who have no hard evidence against him. They offer to set him free if he agrees to testify against his accomplice, who is being held separately by the police, and who is given the same offer. Without knowing what his accomplice will do, should the criminal testify?
- An entrepreneur is trying to decide whether to start a business in a difficult economy. If many other people also start businesses at the same time, their collective spending will ensure strong profits for everyone. Should the entrepreneur start a new business?

All of these problems share several elements: They involve several agents, each with her own goal. The agents' joint behavior determines whether those goals are achieved. In some cases (like chess), an agent can achieve her goals only at the expense of the other agents, while in other cases (like an economy), many agents can achieve

good outcomes. The study of how agents behave in these kinds of situations is the subject of *game theory*.

In this chapter, we focus on a particular kind of game, called a *two-player zero-sum* game. As the name suggests, these kinds of games have only two agents, or *players*. They are called "zero-sum" because the two players have diametrically opposing goals. The behavior of a player in the game is called her *strategy*. We are concerned with determining the best, or *optimal*, strategy for each player in a zero-sum game. If there is a recurring theme in game-theoretic literature, it is this: the appropriate definition of an optimal strategy is a very subtle issue, because the consequences of one player's strategy depend strongly on which strategies the other players are using. We describe one possible definition of optimality in this chapter, while Chapter 3 is largely devoted to exploring a refinement of this definition.

In Section 2.1 we introduce the basics of the theory of two-player zero-sum games: their representation, the objectives of the players, and main results. We state von Neumann's famous minimax theorem, a result which founded the field of game theory.

In Section 2.2, we describe an alternate representation of two-player zero-sum games, called the *extensive form*, which is more natural for many classes of games, including familiar board games like chess. Our discussion of the extensive form will also give us an opportunity to describe a large class of games for which there are "pure" optimal strategies, an idea we will return to in Chapters 3 and 5.

We go on in Section 2.3 to describe a particular two-player zero-sum game inspired by a problem from machine learning. This game establishes the connection between machine learning and game theory, a connection we will extend in Chapter 5.

In Section 2.4 we describe algorithms for computing optimal strategies in two-player zero-sum games. One of these, called the *multiplicative weights* algorithm, we will describe in considerable detail, as it features prominently in Chapters 3 and 5, and serves as the basis for many of our algorithms.

The study of two-player zero-sum games is part of a vast literature on game theory. The earliest book-length treatment is by von Neumann and Morgenstern [138]. Other classical references are by Dresher [22], Myerson [84] and Owen [92].

2.1 Basic Theory

Perhaps the simplest examples of a two-player zero-sum game is the popular children's pastime "Rock-Paper-Scissors". In this game, each player chooses, without the other player's knowledge, one of three possible *moves*: Rock, Paper, or Scissors. After the moves have been selected, they are revealed, and the winner is determined according to simple set of rules that have a playful interpretation: Rock "breaks" Scissors, Scissors "cuts" Paper, and Paper "covers" Rock. If the same move is played by both players, the game is declared a draw. To allow for a more interesting analysis, one usually assumes that each player chooses her move according to the following two-step process: First, she chooses a distribution over the moves, and then she selects a move randomly according to that distribution.

One way to explicitly describe the rules of this game is with a *game matrix*, as shown in Figure 2.1. The two players are called the *row player* and *column player* respectively. Each entry of the matrix indicates the outcome of the game for a pair of moves — a "0" means that the row player wins, a "1" means that the column player wins, and a "1/2" indicates a draw.

	Column player		
Row player	Rock	Paper	Scissors
Rock	1/2	1	0
Paper	0	1/2	1
Scissors	1	0	1/2

Figure 2.1:
Game matrix for Rock-Paper-Scissors

Formally, a *two-player zero-sum game* is defined by an $n \times m$ matrix $\mathbf{M}$ with entries in $[0, 1]$. A *strategy* for the row player is a distribution on the rows of $\mathbf{M}$, and a strategy for the column player is a distribution on the columns of $\mathbf{M}$. A distribution concentrated on a single row or column is called a *pure strategy*; if this is not necessarily the case, then it is called a *mixed strategy*. In the game, each player chooses a strategy independently, and for any pair of strategies $\mathbf{p}$ and $\mathbf{q}$, the *payoff* of the game is $\mathbf{pMq}$ (where the distributions $\mathbf{p}$ and $\mathbf{q}$ have been written as vectors). Clearly, the payoff is just the expected value of an entry in the matrix $\mathbf{M}$ if the row index is chosen according to $\mathbf{p}$ and the column index is chosen according to $\mathbf{q}$. The row player desires a small payoff, while the column player desires a large payoff — this is what makes the game "zero-sum".

Let us examine the game matrix in Figure 2.1 from the perspective of the row player. Clearly, due to the circular nature of the game, no strategy can guarantee a win. So how should the row player choose a strategy? One approach she can take is to make certain assumptions about how the column player will behave. For example, if these two players have played Rock-Paper-Scissors many times before, and the column player played Scissors most often during these earlier games, then the row player can exploit this tendency in the current game by choosing Rock. In general, if the row player has any prior knowledge about the column player's behavior, then that knowledge can be used to inform the row player's choice of strategy. Of course, this approach presupposes that such prior knowledge is available, and that the column player will continue to play in a predictable manner.

By contrast, the game-theoretic approach to this problem is to *assume the opponent will be perfectly adversarial*. That is, the row player imagines that the column player is endowed with foresight about which strategy the row player will choose[1],

[1]It is important to realize that the column player's foresight extends only to the row player's strategy, i.e. the distribution over rows selected by the row player. The column player is *not* presumed to be able to predict which move will be drawn from this distribution. That knowledge would obviously be much more powerful.

and will always choose a strategy that does as well as possible against it. This optimal "counter-strategy" is called a *best response*. The main advantage of the game-theoretic approach is that it has no risk of making overly optimistic assumptions about the column player's behavior.

Under the assumption that the column player will choose a best response, the row player will want to choose a strategy that achieves the following objective:

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{pMq} \tag{2.1}$$

where $\mathbb{P}$ is the set of all row strategies, and $\mathbb{Q}$ is the set of all column strategies.

By inspecting the matrix in Figure 2.1, it is easy to see that the minimum in (2.1) is realized by the uniform distribution on all the rows of $\mathbf{M}$. Let $\mathbf{p}^*$ be this strategy. For the game described by Figure 2.1 we have

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{pMq} = \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}^*\mathbf{Mq} = 1/2$$

In other words, if the row player chooses the strategy $\mathbf{p}^*$, then, under the assumption that the column player will choose a best response, the expected outcome of the game is a draw, and no strategy can guarantee a better outcome for the row player.

Now let us examine the same game from the perspective of the column player, and also assume that her opponent will choose a best response. Then the column player will want to choose a strategy that achieves the following objective.

$$\max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{pMq} \tag{2.2}$$

Again, it is easy to see that the maximum in (2.2) is realized by the uniform distri-

bution on the columns of $\mathbf{M}$. Let $\mathbf{q}^*$ be this strategy. We have

$$\max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{pMq} = \min_{\mathbf{p}\in\mathbb{P}} \mathbf{pMq}^* = 1/2$$

We have shown that, in the Rock-Paper-Scissors game, the following equality holds:

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{pMq} = \max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{pMq} \tag{2.3}$$

Based on the preceding discussion, this equality should seem somewhat surprising. To review, we examined the game from two perspectives. First, we supposed that the row player must choose a strategy before the column player, who gets to choose her strategy with knowledge of the row player's choice. Then we reversed the situation, and supposed that the row player chooses a strategy with knowledge of the column player's choice. One might expect that the second situation would be better for the row player (and worse for the column player) than the first situation, but in this case, as (2.3) indicates, it did not matter.

Remarkably, the equality in (2.3) holds for *every* game matrix $\mathbf{M}$. The common value of both sides of the equality is called the *game value*, and is denoted v^* (so in the case of Rock-Paper-Scissors, $v^* = 1/2$). A row strategy $\mathbf{p}^*$ that realizes the minimum in (2.3) is called a *minimax strategy*, while a column strategy $\mathbf{q}^*$ that realizes the maximum in (2.3) is called a *maximin strategy*; both strategies are called *optimal strategies*. Clearly we have $\mathbf{p}^*\mathbf{Mq} \leq v^*$ for all column strategies $\mathbf{q} \in \mathbb{Q}$, and $\mathbf{pMq}^* \geq v^*$ for all row strategies $\mathbf{p} \in \mathbb{P}$.

Equation (2.3) is known as the *minimax theorem* [137], and is the most fundamental result in the theory of two-player zero-sum games. In Chapter 3, we will repeatedly make use of the minimax theorem in order to obtain our results. We prove the minimax theorem in Section 2.4.3.

A two-player zero-sum game is a special case of an *n-player noncooperative game*,

and the minimax theorem is special case of the fundamental result due to Nash [86], which proved that every noncooperative n-player game has at least one *Nash equilibrium*. The universal existence of Nash equilibria has made this concept enormously influential, particularly in the field of economics.[2]

2.2 Normal Form versus Extensive Form

Rock-Paper-Scissors is a well-known game, and serves as a useful illustration of basic game-theoretic concepts. But in some ways it is quite unusual. Each player acts just once, when choosing a strategy at the start of the game, and the game's outcome is completely determined by those initial choices.

On the other hand, many familiar recreational games, such as chess, go, and poker, proceed sequentially, with the players alternating turns: one player makes a move, which changes the state of the game, then the other player responds, and the process repeats until a state is reached that determines the game's outcome.

These kinds of games are most naturally represented as *extensive (or tree) form* games, while the matrix-based representation described in the previous section is called the *normal form*. Figure 2.2 illustrates the game of chess in extensive form. Each node of the tree represents a possible state of the game, uniquely determined by the sequence of moves made by both players to that point in the game. There is an arc from node x to node y if it is possible for the player whose turn it is in node x to make a single move and change the state of the game into node y. Each player in the game is arbitrarily assigned the role of either the "minimizing" or "maximizing" player. The leaves of the tree are the final states of the game, labeled with the appropriate payoffs: -1 to indicate that the minimizing player wins, $+1$ to indicate that the maximizing player wins, or 0 to denote a draw. A game of chess can be

[2]Nash's result relies heavily on Kakutani's fixed point theorem [53]. When told of Nash's result, von Neumann famously dismissed it, saying "That's trivial, you know. That's just a fixed point theorem." [85]

viewed as walk from the root of this (enormous) tree to one of its leaves, with the players alternating the decision of which node to move to next.

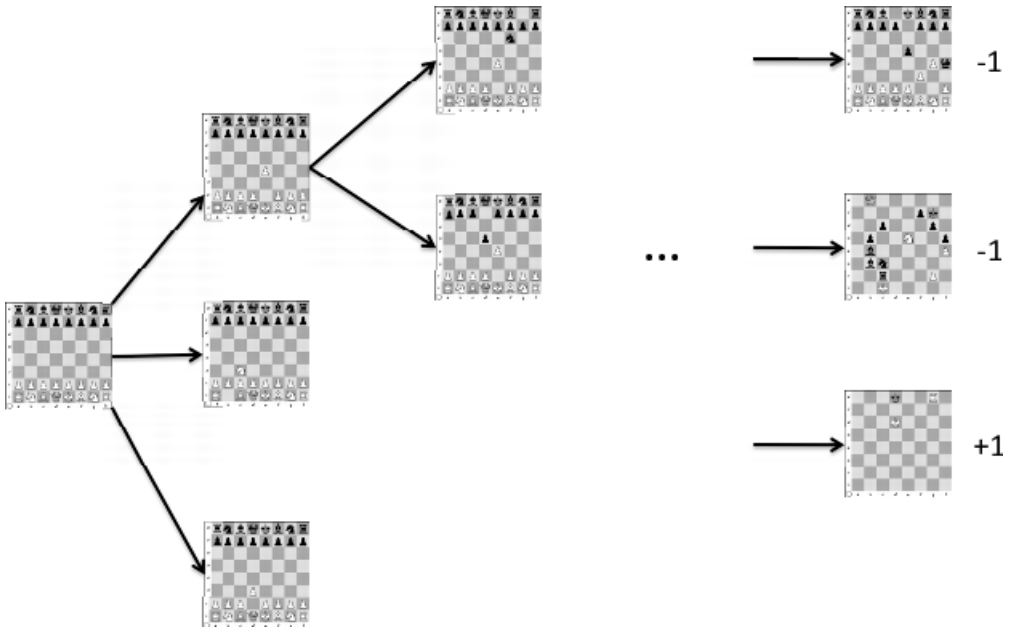

Figure 2.2:
Extensive form representation of chess. (This is just a portion of the entire game tree.)

Chess is a game of *perfect information*, which simply means that both players are completely aware of everything that occurs in the game. Many games cannot be described this way — poker is an example, since each player is unaware of her opponent's cards. Such games are called *imperfect information* games, and the extensive form can be modified to represent them as well, essentially by grouping together nodes of the tree that a player cannot distinguish.

Despite their apparent differences, every extensive form game can be converted into an equivalent normal form game, via a procedure we will now sketch for the case of perfect information games. The key step is to have the players choose *all their moves at the start of the game*, rather than as the game goes along. This is done by having each player choose a *move function* at the start of the game that maps each node in the game tree to a move for that node. Note that, at least theoretically, this

is without loss of generality; it makes no difference to have the players decide far in advance which move they would prefer to make in each situation, including situations that may never actually arise during the course of the game.

The game matrix $\mathbf{M}$ in the resulting normal form representation has one row for each possible move function for the minimizing player, and one column for each possible move function for the maximizing player. The value of $M(i, j)$ is the payoff of the game if the minimizing player chooses her moves according to her ith move function, and the maximizing player chooses her moves according to her jth move function.

Note that each move function is a pure strategy in the game defined by the matrix $\mathbf{M}$, and in normal form games we generally allow players to choose mixed strategies, which in this case are distributions over move functions. This may seem like more power than is really necessary, and in fact it is, as the next theorem attests.

Theorem 2.1. *In every extensive form game of perfect information, each player has a pure optimal strategy.*

Proof. The technique used to prove this result is called "backwards induction". We will construct a pure optimal strategy for the minimizing player; the other case is identical. Consider a node x where it is the minimizing players turn to move, and let $y_1, \ldots, y_k$ be the children of x. Let G_x be the extensive form game defined by the subtree rooted at node x, and define $G_{y_1}, \ldots, G_{y_k}$ similarly. Suppose we have shown that, for all $i \in \{1, \ldots, k\}$, the minimizing player in G_{y_i} has a pure optimal strategy (this is clearly true if $y_1, \ldots, y_k$ are all leaves). Then a pure optimal strategy for the minimizing player in G_x is one that moves, in node x, to the node y_i such that the value of the game G_{y_i} is smallest, and thereafter follows a pure optimal strategy for G_{y_i}. □

Theorem 2.1 tells us that, for a large family of games, pure optimal strategies

always exist. In Section 3.6, we describe an algorithm that can, in some cases, find an optimal strategy that is pure, even in very large games. In Section 5.5.1, we describe a situation where a pure optimal strategy is preferred.

Theorem 2.1 is one of the earliest results in game theory, although its origins are a bit mysterious. This result is almost universally attributed to Zermelo [149], but Schwalbe and Walker [116] point out that this attribution is incorrect, and that Zermelo proved no such thing. (The most probable explanation for the confusion is that Zermelo's original paper was written in German.)

2.3 Example: A Classification Game

Thus far, our examples of games have all literally been "games", the kind people play for amusement. While these kinds of games represent an interesting and non-trivial research area, game theory has far broader applicability. One of the main themes of Chapter 5 will be to explore and extend the connection between game theory and reinforcement learning. To illustrate how such a connection can arise, in this section we will describe how a well-known problem in machine learning can be reduced to finding an optimal strategy in a certain game. This reduction was first described by Freund and Schapire [30].

The problem of *classification* is one of the most well-studied in machine learning. Consider the following archetypical instance of this problem: classifying emails as spam or non-spam. The goal is to find a binary-valued function H, called a *hypothesis*, that accurately classifies emails. Ideally, given an email x, we want $H(x) = +1$ if x is spam, and $H(x) = -1$ otherwise.

When designing a learning algorithm for this problem, the first step is to choose a form for the hypothesis H. A common choice is to let H be a *weighted combination* of a set of binary-valued *base hypotheses* $\mathcal{H}$. For example, we may have a base hypothesis

$h_w \in \mathcal{H}$ for every word w, where $h_w(x) = +1$ if email x contains the word w, and $h_w(x) = -1$ otherwise. A hypothesis $H_{\mathbf{q}}$ is a *weighted combination* of $\mathcal{H}$ if it classifies an email by taking a weighted majority vote, with respect to a distribution $\mathbf{q}$, of all the base hypotheses in $\mathcal{H}$. More formally

$$H_q(x) = \text{sign}\left(\sum_{h \in \mathcal{H}} q(h)h(x)\right) = \text{sign}\left(E_{h \sim \mathbf{q}}[h(x)]\right)$$

Intuitively, a good distribution $\mathbf{q}$ will assign more weight to words that are highly correlated with spam (e.g. "viagra"), and less weight to words that are highly correlated with non-spam (e.g. the names of your friends).

In a typical email classification scenario, a learning algorithm is given a *training set* $\mathcal{X}$ of emails, in which each email is called an *example*, that have all been manually labeled as spam or non-spam. The learning algorithm uses this training set to help it choose a particular weighted hypothesis $H_{\mathbf{q}}$. For example, under certain reasonable assumptions about how the training set was obtained, a hypothesis $H_{\mathbf{q}}$ that has small error on the training set will have small error on the set of *all* emails, including those not in the training set [57]. And this error, called the *generalization error*, is the performance measure we actually care about.

There are other proxies for generalization error besides training error. Define the *margin* $\mu(H_{\mathbf{q}}, x)$ of a hypothesis $H_{\mathbf{q}}$ on an example $x \in \mathcal{X}$ to be

$$\mu(H_{\mathbf{q}}, x) = \left(\sum_{h \in \mathcal{H}} q(h)h(x)\right) \cdot c(x) = E_{h \sim \mathbf{q}}[h(x)] \cdot c(x)$$

where $c(x) \in \{-1, +1\}$ is the correct labeling of the example x. Note that the sign of the margin indicates whether the example was correctly classified by $H_{\mathbf{q}}$ (positive if the classification was correct, and negative otherwise), while the magnitude of the margin basically reflects the confidence $H_{\mathbf{q}}$ has in its classification. It is well-known that a hypothesis for which the *minimum* margin over the training set is as large as

possible will have small generalization error [113].

It turns out that a largest-minimum-margin hypothesis corresponds to a maximin strategy in a certain game. The game is defined by a matrix $\mathbf{M}$, with rows corresponding to training set examples and columns corresponding to base hypotheses, as follows:

$$M(x,h) = \begin{cases} 1 & \text{if } h(x) = c(x) \\ 0 & \text{otherwise.} \end{cases}$$

A maximin strategy $\mathbf{q}^*$ for the column player in this game has the following property:

$$\begin{aligned} \max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{pMq} = \min_{\mathbf{p}} \mathbf{pMq}^* &= \min_{\mathbf{p}} E_{x\sim\mathbf{p},h\sim\mathbf{q}^*}[\mathbf{1}\{h(x) = c(x)\}] \\ &= \min_{x\in\mathcal{X}} E_{h\sim\mathbf{q}^*}[\mathbf{1}\{h(x) = c(x)\}] \end{aligned} \tag{2.4}$$

where $\mathbf{1}\{\cdot\} \in \{0,1\}$ is the indicator function, and where the last equality holds because the minimum is realized by a distribution concentrated on a single example. Note that the quantity in (2.4) is the minimum margin of the hypothesis $H_{\mathbf{q}^*}$ on the training set $\mathcal{X}$, shifted and scaled to lie in the interval $[0,1]$ instead of $[-1,+1]$.

So in this game, the column player's strategy $\mathbf{q}$ specifies a hypothesis $H_{\mathbf{q}}$, which in turn determines a margin for each example in the training set. The row player's strategy $\mathbf{p}$ specifies a distribution on the examples. The game's payoff is the average margin of $H_{\mathbf{q}}$ with respect to $\mathbf{p}$. Thus, because $\mathbf{q}^*$ is a maximin strategy, the hypothesis $H_{\mathbf{q}^*}$ has the largest minimum margin on the training set of any hypothesis that is a weighted combination of $\mathcal{H}$.

2.4 Algorithms for Solving Games

A topic of great historical interest in game theory, and one that is major focus of Chapters 3 and 5, is computing the optimal strategies for the row and column player in a game. This is also called *solving* the game. We will describe several well-known methods for solving games in the next few sections. One of the methods, the multiplicative weights algorithm, will form the basis of the algorithms we develop later in this thesis. The key feature of the multiplicative weights algorithm is that it can solve games in which one player has an enormous (even infinite) number of pure strategies. In Chapter 5, we explain how this makes it especially well-suited for an application to a certain reinforcement learning problem.

2.4.1 Linear Programming

Perhaps the most obvious method for solving a game is via a technique called *linear programming*. Many of the best algorithms for solving games are based on linear programming, including some that we will discuss in Chapters 3 and 5. Consider the following optimization problem.

$$\max_{\mathbf{x}} \; F(\mathbf{x}) \tag{2.5}$$

$$\mathbf{x} \in \mathcal{X} \tag{2.6}$$

If F is linear function, and if $\mathcal{X}$ can be described by a set of linear equalities and inequalities, then we say that (2.5)-(2.6) is a *linear program*. The study of linear programs is a very active research area, and many efficient solution methods have been developed. The ellipsoid algorithm [60] was the first method for finding an $\mathbf{x}^*$ that solves the linear program (2.5)-(2.6) in time polynomial in the number of bits needed to specify F and $\mathcal{X}$. Unfortunately in practice the ellipsoid algorithm is quite

inefficient. More recently, interior point algorithms [55, 81] have been developed that share the polynomial-time performance guarantees of the ellipsoid algorithm and also exhibit dramatically better efficiency in practice. An early algorithm called the simplex method [17] is also very efficient, and is quite simple to implement, but requires exponential time in the worst-case [61].

Solving a game is simply a matter of setting up an appropriate linear program. Let us begin by considering the problem of computing an optimal strategy for the row player. We need to find a row strategy $\mathbf{p}^*$ that realizes the minimum in (2.1). The key observation is that the inner maximum in (2.1) is realized by a pure column strategy, because with respect to this maximization the row strategy $\mathbf{p}$ has already been fixed. In other words, we have

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{pMq} = \min_{\mathbf{p}\in\mathbb{P}} \max_{j} M(\mathbf{p}, j)$$

This can be equivalently written as

$$\min_{\mathbf{p}} \max_{j} M(\mathbf{p}, j) \tag{2.7}$$

$$p(i) \geq 0 \text{ for } i = 1 \ldots n \tag{2.8}$$

$$\sum_{i=1}^{n} p(i) = 1 \tag{2.9}$$

where the constraints (2.8)-(2.9) have been introduced to specify the set $\mathbb{P}$. We can

also introduce a variable v to represent the value of the inner maximum.

$$\min_{\mathbf{p},v} \; v \tag{2.10}$$

$$v \geq M(\mathbf{p}, j) \text{ for } j = 1 \ldots m \tag{2.11}$$

$$p(i) \geq 0 \text{ for } i = 1 \ldots n \tag{2.12}$$

$$\sum_{i=1}^{n} p(i) = 1 \tag{2.13}$$

Clearly, the optimization problem (2.10)-(2.13) is a linear program. Its solution $(\mathbf{p}^*, v^*)$ contains an optimal row strategy $\mathbf{p}^*$ for the row player, along with an upper bound v^* on the payoff of the game, assuming that the row player chooses strategy $\mathbf{p}^*$.

A nearly identical argument can be made on behalf of the column player, yielding the following linear program for computing an optimal column strategy $\mathbf{q}^*$.

$$\max_{\mathbf{q},v} \; v \tag{2.14}$$

$$v \leq M(i, \mathbf{q}) \text{ for } i = 1 \ldots n \tag{2.15}$$

$$q(j) \geq 0 \text{ for } j = 1 \ldots m \tag{2.16}$$

$$\sum_{j=1}^{m} q(i) = 1 \tag{2.17}$$

Readers knowledgeable in a concept called *LP duality* [12] will observe that the linear program in (2.10)-(2.13) is the dual of the linear program in (2.14)-(2.17). This immediately proves that the minimum in (2.10) is equal to the maximum in (2.14), and thus proves the minimax theorem. However, instead of entering into a discussion of LP duality, we will prove the minimax theorem via a different argument in Section 2.4.3.

2.4.2 Fictitious Play

The next algorithm we will describe for solving games is most naturally presented in the context of a *repeated game*. Like the *one-shot* games that we have studied so far, a repeated game is defined by a game matrix $\mathbf{M}$. However, unlike a one-shot game, a repeated game proceeds for T rounds. In each round $t = 1 \dots T$

1. Row player chooses row strategy $\mathbf{p}_t$.

2. Column player chooses column strategy $\mathbf{q}_t$ with knowledge of $\mathbf{p}_t$.

Let $\overline{\mathbf{p}}_t = \frac{1}{t}\sum_{t'=1}^{t} \mathbf{p}_{t'}$ and $\overline{\mathbf{q}}_t = \frac{1}{t}\sum_{t'=1}^{t} \mathbf{q}_{t'}$ be the average of the row and column strategies, respectively, over the first t rounds.

Suppose that the strategies $\mathbf{p}_t$ and $\mathbf{q}_t$ in each round t are chosen as follows: $\mathbf{p}_t$ is a best response to $\overline{\mathbf{q}}_{t-1}$, and $\mathbf{q}_t$ is a best responses to $\overline{\mathbf{p}}_t$. One can justify this approach to choosing strategies in the following way. First, suppose each player always chooses a pure strategy. This is without loss of generality, since there is always a best response with this property. Now consider the repeated game from the perspective of the row player (the story for the column player is identical). Suppose the row player is unaware of how the column player actually chooses her strategies, and instead assumes a simple model for her behavior: she assumes that the column player has a single underlying mixed strategy, which is never directly revealed, and that the pure strategy chosen by the column player in each round is an independent draw from this distribution (recall that a mixed strategy is just a distribution on the set of pure strategies). In this case, in each round t, the rational behavior for the row player is to choose a strategy that is a best response to the current maximum likelihood estimate of the column player's mixed strategy, which is $\overline{\mathbf{q}}_{t-1}$.

If the two players choose their strategies in the manner described above, then the repeated game will have the following very desirable property: the *average* of the players' strategies (i.e., $\overline{\mathbf{p}}_T$ and $\overline{\mathbf{q}}_T$) will converge to the optimal strategies of

the one-shot game defined by the matrix $\mathbf{M}$. This algorithm for solving a game is called *fictitious play*, and was first described by Robinson [108], who resolved an earlier conjecture by Brown [15]. The next theorem makes precise the convergence guarantee for fictitious play.

Theorem 2.2 (Robinson [108])**.** *Suppose the fictitious play algorithm is run for T rounds. Let $\overline{\mathbf{p}}_T$ and $\overline{\mathbf{q}}_T$ be the average of all the row and column strategies, respectively, generated during the algorithm. Then for all $\epsilon > 0$*

$$\min_{\mathbf{p} \in \mathbb{P}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T \geq v^* - \epsilon$$

$$\max_{\mathbf{q} \in \mathbb{Q}} \overline{\mathbf{p}}_T \mathbf{M}\mathbf{q} \leq v^* + \epsilon$$

for all sufficiently large T.

Note that, unlike the linear programming approach described in the previous section, the amount of computation required by the fictitious play algorithm does not explicitly depend on the size of the matrix $\mathbf{M}$. All that is required is a subroutine for each player that can compute a best response to any opponent strategy.

2.4.3 Multiplicative Weights Algorithm

The motivation for the fictitious play algorithm described in the previous section may have seemed a little mysterious. We presented fictitious play as a method for choosing strategies in a repeated game, but we never actually specified what the goal of the repeated game was.

So let us define the payoff of the repeated game as the average payoff $\frac{1}{T}\sum_t \mathbf{p}_t\mathbf{M}\mathbf{q}_t$ of all the one-shot games over all T rounds. As usual, the row player will desire a small payoff, while the column player will desire a large payoff. Because we will be primarily analyzing the repeated game from the perspective of the row player in this section, we say the quantity $\mathbf{p}_t\mathbf{M}\mathbf{q}_t$ is the *loss* suffered by the row player in round t.

We are interested in algorithms for choosing $\mathbf{p}_1, \ldots, \mathbf{p}_T$ in the repeated game so that the average loss $\frac{1}{T}\sum_t \mathbf{p}_t \mathbf{M} \mathbf{q}_t$ suffered by the row player over all T rounds is small — in particular, we wish it to be small relative to the least average loss $\min_{\mathbf{p}\in\mathbb{P}} \frac{1}{T}\sum_t \mathbf{p}\mathbf{M}\mathbf{q}_t$ suffered by any *single* row strategy. If the difference between these two losses vanishes as the number rounds goes to infinity, the algorithm is said to be a *no-regret* algorithm [16]. The *multiplicative weights (MW)* algorithm [30] is a particular no-regret algorithm that we will use extensively in this thesis.

As we will see, any no-regret algorithm can be used to solve a game in the usual one-shot setting, i.e., to compute optimal strategies in a two-player zero-sum game. In fact, the well-known AdaBoost algorithm [31] for classification is based on using the MW algorithm to compute a maximin strategy in the classification game from Section 2.3. No-regret algorithms also provide a simple method for proving the minimax theorem. Proving these results is the goal of this section, which will closely follow the development due to Freund and Schapire [32].

The MW algorithm is related to earlier work by Hannan [43], Vovk [139] and Littlestone and Warmuth [71]. This literature initiated the field of *online learning and prediction*. An excellent book-length survey of this topic is by Cesa-Bianchi and Lugosi [16]. Recently, the online learning framework has been generalized to *online convex optimization* by Zinkevich [150] (see also Gordon [38], Kalai and Vempala [54] for earlier similar work).

The Algorithm

In each round t, the MW algorithm maintains a weight $w_t(i)$ for each row i of the matrix $\mathbf{M}$. In round t, the row player chooses a strategy $\mathbf{p}_t$ such that

$$p_t(i) = \frac{w_t(i)}{\sum_{j=1}^{n} w_t(j)}$$

Each weight $w_t(i)$ is calculated as follows: If $t = 1$, then $w_t(i) = \alpha_i$, where $\alpha_i \geq 0$ is a parameter (and thus $\mathbf{p}_1$ is completely determined by the choice of $\alpha_1, \ldots, \alpha_n$). Otherwise

$$w_t(i) = w_{t-1}(i)\beta^{M(i,\mathbf{q}_{t-1})}$$

where $\beta \in [0, 1)$ is a parameter.

The MW algorithm has a nice intuitive interpretation: For any two rows i and j, if the pure strategy concentrated on row i would have suffered less loss in round $t-1$ than the pure strategy concentrated on row j, then in round t the weight on row i is "boosted" relative to the weight on row j.

Notice that the row player actually needs very little information in order to execute the MW algorithm. She does not need to observe the strategy $\mathbf{q}_t$, or even the entire matrix $\mathbf{M}$, directly. She only needs to observe, in each round t, the quantities $M(i, \mathbf{q}_t)$ for $i = 1, \ldots, n$.

Regret Bound

Over the course of the repeated game, the MW algorithm shifts the weight of $\mathbf{p}_t$ onto rows that tend to suffer less loss. So one might hope that the row player's average loss $\frac{1}{T}\sum_t \mathbf{p}_t\mathbf{M}\mathbf{q}_t$ over the entire repeated game will be small. Indeed, we will prove a bound on the following difference, which is known as the *row player's average regret*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t\mathbf{M}\mathbf{q}_t - \min_{\mathbf{p}\in\mathbb{P}} \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}\mathbf{M}\mathbf{q}_t \tag{2.18}$$

This is the difference between the row player's average loss, and the average loss she could have suffered by playing a single row strategy every round.

The bound on the row player's average regret will be a straightforward consequence of the next theorem.

Theorem 2.3. *Suppose the repeated game is played on a matrix* $\mathbf{M}$ *for* T *rounds, and*

the row player chooses strategies $\mathbf{p}_1, \ldots, \mathbf{p}_T$ *according to the MW algorithm. Then*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t \mathbf{M} \mathbf{q}_t \leq \frac{\log(1/\beta)}{1-\beta} \min_{\mathbf{p}\in\mathbb{P}} \left[\frac{1}{T}\sum_{t=1}^{T} \mathbf{p}\mathbf{M}\mathbf{q}_t + \frac{\mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}_1)}{T(1-\beta)} \right]$$

where the relative entropy $\mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}')$ *between distributions* $\mathbf{p}$ *and* $\mathbf{p}'$ *is defined*

$$\mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}') = \sum_{i=1}^{n} p(i) \log \frac{p(i)}{p'(i)}$$

Proof. Let $Z_t = \sum_{i=1}^{n} p_t(j')\beta^{M(j',\mathbf{q}_t)}$. We can use Z_t to express $\mathbf{p}_{t+1}$ in terms of $\mathbf{p}_t$. Specifically, for all rows i and rounds t we have

$$\begin{aligned} p_{t+1}(i) = \frac{w_{t+1}(i)}{\sum_{i'=1}^{n} w_{t+1}(i')} = \frac{w_t(i)\beta^{M(i,\mathbf{q}_t)}}{\sum_{i'=1}^{n} w_t(i')\beta^{M(i',\mathbf{q}_t)}} &= \frac{p_t(i)\beta^{M(i,\mathbf{q}_t)}}{\sum_{i'=1}^{n} p_t(i')\beta^{M(i',\mathbf{q}_t)}} \\ &= p_t(i)\frac{\beta^{M(i,\mathbf{q}_t)}}{Z_t} \end{aligned} \tag{2.19}$$

For any round t of the MW algorithm and any $\mathbf{p} \in \mathbb{P}$ we have

$$\begin{aligned}
&\mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}_{t+1}) - \mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}_t) \\
&= \sum_{i=1}^{n} p(i) \log \frac{p(i)}{p_{t+1}(i)} - \sum_{i=1}^{n} p(i) \log \frac{p(i)}{p_t(i)} \\
&= \sum_{i=1}^{n} p(i) \log \frac{p_t(i)}{p_{t+1}(i)} \\
&= \sum_{i=1}^{n} p(i) \log \frac{Z_t}{\beta^{M(i,\mathbf{q}_t)}} \\
&= \log\left(\frac{1}{\beta}\right) \sum_{i=1}^{n} p(i) M(i, \mathbf{q}_t) + \log Z_t \\
&= \log\left(\frac{1}{\beta}\right) \mathbf{p}\mathbf{M}\mathbf{q}_t + \log\left(\sum_{i=1}^{n} p_t(i) \beta^{M(i,\mathbf{q}_t)}\right) \\
&\leq \log\left(\frac{1}{\beta}\right) \mathbf{p}\mathbf{M}\mathbf{q}_t + \log\left(\sum_{i=1}^{n} p_t(i) \left(1 - (1-\beta) M(i, \mathbf{q}_t)\right)\right) \qquad (2.20) \\
&= \log\left(\frac{1}{\beta}\right) \mathbf{p}\mathbf{M}\mathbf{q}_t + \log\left(1 - (1-\beta)\mathbf{p}_t\mathbf{M}\mathbf{q}_t\right) \\
&\leq \log\left(\frac{1}{\beta}\right) \mathbf{p}\mathbf{M}\mathbf{q}_t - (1-\beta)\mathbf{p}_t\mathbf{M}\mathbf{q}_t \qquad (2.21)
\end{aligned}$$

In (2.20) we used the fact that, by convexity, $\beta^x \leq 1 - (1-\beta)x$ for $\beta \geq 0$ and $x \in [0, 1]$. In (2.21) we used the fact that $\log(1-x) \leq -x$ for $x < 1$.

Summing this inequality over all $1 \leq t \leq T$ yields

$$\mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}_{T+1}) - \mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}_1) \leq \left(\log \frac{1}{\beta}\right) \sum_{t=1}^{T} \mathbf{p}\mathbf{M}\mathbf{q}_t - (1-\beta) \sum_{t=1}^{T} \mathbf{p}_t\mathbf{M}\mathbf{q}_t$$

Noting that $\mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}_{T+1}) \geq 0$, rearranging the inequality, multiplying both sides by $1/T$, and noting that $\mathbf{p} \in \mathbb{P}$ was chosen arbitrarily yields the theorem. □

The statement of Theorem 2.3 can be simplified somewhat, by converting the multiplicative and additive penalties in the upper bound into a single additive penalty.

Corollary 2.4. *Suppose the repeated game is played on a matrix* $\mathbf{M}$ *for* T *rounds, and*

the row player chooses strategies $\mathbf{p}_1, \ldots, \mathbf{p}_T$ *according to the MW algorithm. Then*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t \mathbf{M} \mathbf{q}_t \leq \frac{1}{T} \min_{\mathbf{p} \in \mathbb{P}} \sum_{t=1}^{T} \mathbf{p} \mathbf{M} \mathbf{q}_t + \Delta_{T, \mathbf{p}_1, \beta}$$

where

$$\Delta_{T, \mathbf{p}_1, \beta} \triangleq \frac{\log(1/\beta) - (1-\beta)}{(1-\beta)} + \max_{\mathbf{p} \in \mathbb{P}} \frac{\mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}_1)}{T(1-\beta)}$$

Proof. By observing that $\sum_{t=1}^{T} \mathbf{p} \mathbf{M} \mathbf{q}_t \leq T$ for all $\mathbf{p} \in \mathbb{P}$, we can easily manipulate the inequality in Theorem 2.3 to prove the theorem. □

The quality of the regret bound in Corollary 2.4 depends on the magnitude of $\Delta_{T, \mathbf{p}_1, \beta}$. The following lemma bounds $\Delta_{T, \mathbf{p}_1, \beta}$ for tuned choices of β and $\alpha_1, \ldots, \alpha_n$.

Lemma 2.5. *If* β *is set to*

$$\frac{1}{1 + \sqrt{\frac{2 \log n}{T}}}$$

and $\alpha_i = \alpha_j$ *for all rows* i *and* j*, then* $\Delta_{T, \mathbf{p}_1, \beta} \leq \Delta_{T,n}$*, where*

$$\Delta_{T,n} \triangleq \sqrt{\frac{2 \log n}{T}} + \frac{\log n}{T}$$

Proof. If $\alpha_i = \alpha_j$ for all rows i and j, then $\mathbf{p}_1$ is the uniform distribution, and thus $\mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}_1) \leq \log n$ for all $\mathbf{p} \in \mathbb{P}$. Also, it can be shown that $-\log \beta \leq (1-\beta^2)/(2\beta)$. Applying this approximation and the given choice of β to the definition of $\Delta_{T, \mathbf{p}_1, \beta}$ yields the result. □

Since $\Delta_{T,n} \to 0$ as $T \to \infty$, by using the MW algorithm the row player's average regret can be made less than any positive constant, assuming the repeated game is sufficiently long. Thus MW is a no-regret algorithm for the row player (an earlier term for this property is *Hannan consistency* [43]).

Proving the Minimax Theorem and Solving a Game

In this section, we will prove the minimax theorem by using the existence of no-regret algorithms for the repeated game. We will also show how the MW algorithm — in fact, any no-regret algorithm — can be used to solve a game.

By analogy to the definition in (2.18), we define the *column player's average regret* in the repeated game as follows

$$\max_{\mathbf{q}\in\mathbb{Q}} \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t\mathbf{M}\mathbf{q} - \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t\mathbf{M}\mathbf{q}_t$$

Because the column player chooses a strategy *after* the row player in each round of the repeated game, a no-regret algorithm for the column player is extremely simple to implement: In each round t, just choose a best response to the row player's strategy, i.e., let $\mathbf{q}_t = \arg\max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}_t\mathbf{M}\mathbf{q}$. Then the column player's average regret is guaranteed to be at most zero in a repeated game of any length T.

We have thus far shown that no-regret algorithms exist for both the row player and column player in the repeated game. The existence of these algorithms will allow us to prove the remaining results of this section, which will be straightforward extensions of the next theorem. For the remainder of this chapter, recall our definition from Section 2.4.2 that $\overline{\mathbf{p}}_T = \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t$ and $\overline{\mathbf{q}}_T = \frac{1}{T}\sum_{t=1}^{T} \mathbf{q}_t$.

Theorem 2.6. *Suppose the repeated game is played on a matrix* $\mathbf{M}$ *for* T *rounds, and the row player's average regret is at most* ϵ_1*, and the column player's average regret is at most* ϵ_2*. Then*

$$\max_{\mathbf{q}\in\mathbb{Q}} \overline{\mathbf{p}}_T\mathbf{M}\mathbf{q} \leq \min_{\mathbf{p}\in\mathbb{P}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T + \epsilon_1 + \epsilon_2$$

Proof. Note that $\overline{\mathbf{p}}_T \in \mathbb{P}$ and $\overline{\mathbf{q}}_T \in \mathbb{Q}$. Therefore

$$\begin{aligned}
&\max_{\mathbf{q}\in\mathbb{Q}} \overline{\mathbf{p}}_T \mathbf{M}\mathbf{q} \\
&= \max_{\mathbf{q}\in\mathbb{Q}} \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t \mathbf{M}\mathbf{q} \\
&\leq \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t \mathbf{M}\mathbf{q}_t + \epsilon_1 \\
&\leq \min_{\mathbf{p}\in\mathbb{P}} \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}\mathbf{M}\mathbf{q}_t + \epsilon_1 + \epsilon_2 \\
&= \min_{\mathbf{p}\in\mathbb{P}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T + \epsilon_1 + \epsilon_2
\end{aligned}$$

where the inequalities above follow from our assumptions about each player's average regret. □

We are now ready to prove the minimax theorem.

Theorem 2.7 (Minimax Theorem)**.** *For any matrix* $\mathbf{M}$

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}\mathbf{M}\mathbf{q} = \max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{p}\mathbf{M}\mathbf{q} \triangleq v^*$$

Proof. The inequality

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}\mathbf{M}\mathbf{q} \geq \max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{p}\mathbf{M}\mathbf{q}$$

is easy to establish, since clearly

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}\mathbf{M}\mathbf{q} = \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}^*\mathbf{M}\mathbf{q} \geq \max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{p}\mathbf{M}\mathbf{q}$$

We claim that

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}\mathbf{M}\mathbf{q} \leq \max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{p}\mathbf{M}\mathbf{q}$$

which proves the theorem. Consider a repeated game in which the row player's average

regret is at most ϵ_1, and the column player's average regret is at most ϵ_2. Then by Theorem 2.6 we have

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{pMq} \leq \max_{\mathbf{q}\in\mathbb{Q}} \overline{\mathbf{p}}_T\mathbf{Mq} \leq \min_{\mathbf{p}\in\mathbb{P}} \mathbf{pM}\overline{\mathbf{q}}_T + \epsilon_1 + \epsilon_2 \leq \max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{pMq} + \epsilon_1 + \epsilon_2$$

By the existence of no-regret algorithms for the row and column players, ϵ_1 and ϵ_2 can be made arbitrarily small for any sufficiently long repeated game, and thus the claim follows. □

Theorem 2.6 also implies that no-regret algorithms can be used to solve a game. The next theorem proves this for the particular case of the MW algorithm, although it will be clear from the proof that any no-regret algorithm will suffice.

Theorem 2.8. *Suppose that the repeated game is played on a matrix* $\mathbf{M}$ *for* T *rounds, and the row player chooses strategies* $\mathbf{p}_1, \ldots, \mathbf{p}_T$ *according to the MW algorithm, and the column player chooses* $\mathbf{q}_t = \arg\max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}_t\mathbf{Mq}$ *in each round* t*. Then*

$$\max_{\mathbf{q}\in\mathbb{Q}} \overline{\mathbf{p}}_T\mathbf{Mq} \leq v^* + \Delta_{T,\mathbf{p}_1,\beta}$$

$$\min_{\mathbf{p}\in\mathbb{P}} \mathbf{pM}\overline{\mathbf{q}}_T \geq v^* - \Delta_{T,\mathbf{p}_1,\beta}$$

Proof. By the way that the row player and column player choose their strategies, we can directly apply Theorem 2.6 with $\epsilon_1 = \Delta_{T,\mathbf{p}_1,\beta}$ and $\epsilon_2 = 0$. Along with the minimax theorem, this yields

$$\max_{\mathbf{q}\in\mathbb{Q}} \overline{\mathbf{p}}_T\mathbf{Mq} \leq \min_{\mathbf{p}\in\mathbb{P}} \mathbf{pM}\overline{\mathbf{q}}_T + \Delta_{T,\mathbf{p}_1,\beta} \leq \max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{pMq} + \Delta_{T,\mathbf{p}_1,\beta} = v^* + \Delta_{T,\mathbf{p}_1,\beta}$$

and

$$\min_{\mathbf{p}\in\mathbb{P}} \mathbf{pM}\overline{\mathbf{q}}_T \geq \max_{\mathbf{q}\in\mathbb{Q}} \overline{\mathbf{p}}_T\mathbf{Mq} - \Delta_{T,\mathbf{p}_1,\beta} \geq \min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{pMq} - \Delta_{T,\mathbf{p}_1,\beta} = v^* - \Delta_{T,\mathbf{p}_1,\beta}$$

□

As a method for computing optimal strategies, the MW algorithm can be seen as method whose requirements are a hybrid of the requirements of linear programming and fictitious play. The linear programming approach requires time and space polynomial in nm, the size of the game matrix $\mathbf{M}$. Fictitious play has no explicit dependence on the size of $\mathbf{M}$, but does require best-response oracles for both the row and column players. Fictitious play also lacks a non-asymptotic convergence guarantee. In comparison, the MW algorithm requires time and space polynomial in only n (the number of rows of $\mathbf{M}$), a best-response oracle for just the column player, and has an explicit bound on its convergence time. So the MW algorithm is especially well-designed for game matrices with a prohibitively large number of columns, and for which an efficient best-response oracle for the column player exists. As we will see in Chapter 5, the games that we are interested in solving will have exactly these properties.

2.5 Restricted Games

In Chapter 3, we will consider strategies that are not only optimal for the game matrix $\mathbf{M}$, but are also optimal for certain "subgames" defined by $\mathbf{M}$. As a result, we will need to examine *restricted games*, where the row and column players' strategies are restricted to subsets $\mathcal{P} \subset \mathbb{P}$ and $\mathcal{Q} \subseteq \mathbb{Q}$, respectively. When $\mathcal{P}$ and $\mathcal{Q}$ are closed and convex, essentially all the results from Section 2.4.3 carry over to the restricted setting.

Define the *row player's average $\mathcal{P}$-regret* to be

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}_t\mathbf{M}\mathbf{q}_t - \min_{\mathbf{p}\in\mathcal{P}}\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}\mathbf{M}\mathbf{q}_t$$

and similarly define the *column player's average $\mathcal{Q}$-regret* to be

$$\max_{\mathbf{q}\in\mathcal{Q}} \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t\mathbf{M}\mathbf{q} - \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t\mathbf{M}\mathbf{q}_t$$

Obviously, the row and column player's average regret are, respectively, special cases of average $\mathcal{P}$-regret and average $\mathcal{Q}$-regret, where $\mathcal{P} = \mathbb{P}$ and $\mathcal{Q} = \mathbb{Q}$.

Theorem 2.9. *Suppose the repeated game is played on a matrix $\mathbf{M}$ for T rounds. Suppose $\mathbf{p}_1, \ldots, \mathbf{p}_T \in \mathcal{P}$ and $\mathbf{q}_1, \ldots, \mathbf{q}_T \in \mathcal{Q}$, where $\mathcal{P} \subseteq \mathbb{P}$ and $\mathcal{Q} \subseteq \mathbb{Q}$ are closed and convex. Suppose the row player's average $\mathcal{P}$-regret is at most ϵ_1, and the column player's average $\mathcal{Q}$-regret is at most ϵ_2. Then*

$$\max_{\mathbf{q}\in\mathcal{Q}} \overline{\mathbf{p}}_T\mathbf{M}\mathbf{q} \leq \min_{\mathbf{p}\in\mathcal{P}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T + \epsilon_1 + \epsilon_2$$

Moreover, if $\epsilon_1 \to 0$ and $\epsilon_2 \to 0$ as $T \to \infty$, then

$$\min_{\mathbf{p}\in\mathcal{P}} \max_{\mathbf{q}\in\mathcal{Q}} \mathbf{p}\mathbf{M}\mathbf{q} = \max_{\mathbf{q}\in\mathcal{Q}} \min_{\mathbf{p}\in\mathcal{P}} \mathbf{p}\mathbf{M}\mathbf{q} \triangleq v^*_{\mathcal{P},\mathcal{Q}}$$

and

$$\max_{\mathbf{q}\in\mathbb{Q}} \overline{\mathbf{p}}_T\mathbf{M}\mathbf{q} \leq v^*_{\mathcal{P},\mathcal{Q}} + \epsilon_1 + \epsilon_2$$

$$\min_{\mathbf{p}\in\mathbb{P}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T \geq v^*_{\mathcal{P},\mathcal{Q}} - \epsilon_1 - \epsilon_2$$

Proof. The minima and maxima exist because $\mathcal{P}$ and $\mathcal{Q}$ are closed and bounded (they are bounded because $\mathbb{P}$ and $\mathbb{Q}$ are bounded). Also, by convexity, we have $\overline{\mathbf{p}}_T \in \mathcal{P}$ and $\overline{\mathbf{q}}_T \in \mathcal{Q}$. The remainder of the proof is essentially identical to the proofs of Theorems 2.6-2.8. □

As before, it is easy to implement an algorithm for the column player that has average $\mathcal{Q}$-regret at most zero: Simply let $\mathbf{q}_t = \arg\max_{\mathbf{q}\in\mathcal{Q}} \mathbf{p}_t\mathbf{M}\mathbf{q}$ in each round t.

But is there an algorithm for the row player for which average $\mathcal{P}$-regret approaches zero as T becomes large? Note that we cannot necessarily use the MW algorithm, because we will not be guaranteed that $\mathbf{p}_t, \ldots, \mathbf{p}_T \in \mathcal{P}$ for an arbitrary closed and convex $\mathcal{P} \in \mathbb{P}$, and so Theorem 2.9 will no longer necessarily be true.

One common approach to circumventing this problem is to *project* the row strategies onto the set $\mathcal{P}$ in each round [16]. There is a family of subsets of $\mathbb{P}$ for which this is particularly easy to do: Let $I \subseteq [n]$ be a subset of row indices of the matrix $\mathbf{M}$, and let

$$\mathbb{P}_I \triangleq \{\mathbf{p} \in \mathbb{P} : p(i) = 0 \text{ for } i \notin I\}$$

be the set of all row strategies whose support is contained in I. Clearly $\mathbb{P}_I$ is closed and convex for all $I \subseteq [n]$.

There is a natural operation by which a row strategy $\mathbf{p}_t$ can be projected onto the set $\mathbb{P}_I$. Define $\mathbf{p}_{I,t}$ as follows.

$$p_{I,t}(j) \triangleq \begin{cases} p_t(j)/Z_{I,t} & j \in I \\ 0 & j \notin I \end{cases}$$

where $Z_{I,t} \triangleq \sum_{j \in I} p_t(j)$ is a normalization constant ensuring that $\mathbf{p}_{I,t}$ is a distribution. Clearly $\mathbf{p}_{I,t} \in \mathbb{P}_I$.

Theorem 2.10. *Suppose the repeated game is played on a matrix* $\mathbf{M}$ *for* T *rounds, and the row player chooses strategies* $\mathbf{p}_1, \ldots, \mathbf{p}_T$ *according to the MW algorithm. Let* $I \subseteq [n]$. *Then*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_{I,t}\mathbf{M}\mathbf{q}_t \leq \frac{1}{T}\min_{\mathbf{p} \in \mathbb{P}_I}\sum_{t=1}^{T} \mathbf{p}\mathbf{M}\mathbf{q}_t + \Delta_{T,\mathbf{p}_1,\beta}$$

Proof. Let $\mathbf{M}_I$ be a matrix identical to $\mathbf{M}$, except that all the rows not contained in I are removed. Observe that the row strategies $\mathbf{p}_{I,t}$ are the same as those produced by the MW algorithm if it is run on the matrix $\mathbf{M}_I$. Thus Corollary 2.4 immediately

implies that the row player's regret is at most

$$\frac{\log(1/\beta) - (1-\beta)}{(1-\beta)} + \max_{\mathbf{p} \in \mathbb{P}_I} \frac{\mathrm{RE}(\mathbf{p} \mid\mid \mathbf{p}_{I,1})}{T(1-\beta)}$$

It is easy to verify that this quantity is upper bounded by $\Delta_{T,\mathbf{p}_1,\beta}$. □

2.6 Other Related Work

We have seen that no-regret algorithms converge to the set of equilibria of zero-sum games. Variations on the definition of regret can be used to prove convergence to other equilibria. One generalization of regret is called *internal (or swap) regret* [29], where the performance of an algorithm is compared to the best "swapping" of its choices in hindsight. Hart and Mas-Colell [44] showed that, in n-player repeated games, no-internal-regret algorithms converge to the set of *correlated equilibria*, which includes the set of all Nash equilibria. Greenwald and Jafari [42] generalized the notion of internal regret to a much larger class of transformations, and showed that algorithms which have no regret in this more general sense converge to a set of similarly-defined equilibria. There has been considerable additional research in this vein [122, 46, 39].

The algorithms for solving games presented in this chapter operate on the normal form representation. For some games, the extensive form is more compact, and there are many algorithms designed for this setting. Koller et al. [63] gave the first practical algorithm for computing equilibria of games of incomplete information that is linear in size of extensive form representation. Zinkevich et al. [151] compute minimax/maximin strategies in very large extensive form zero-sum games by using best-response oracles. The complexity of their algorithm depends on a quantity called the *range of skill* of the game, which may be less than size of normal form representation, though there is no guarantee of this. Zinkevich et al. [152] use regret minimization to compute equilibria in large extensive form zero-sum games with incomplete informa-

tion. They minimize a quantity called *counterfactual regret*, which is an upper-bound on regret.

Some games with considerable structure have representations which are even more compact than the normal or extensive form, and which lend themselves to even more efficient solution techniques. Perhaps the most well-known example of this kind of representation is a *graphical game* [59], which exploits independencies in the interactions of the players. More recently, this framework has been extended to *local effect games* by Leyton-Brown and Tennenholtz [70] and *action-graph games* by Bhat and Leyton-Brown [9], which model context-specific independencies.

2.7 Conclusion

In this chapter, we discussed the basics of the theory of two-player zero-sum games. We gave the standard definition of optimal behavior in these kinds of games, and described the main ways that they can be represented. We also presented techniques for solving zero-sum games, including no-regret algorithms, which will be the focus of Chapters 3 and 5.

Chapter 3

Lexicographic Optimal Strategies

In Chapter 2, we defined an optimal strategy in a two-player zero-sum game as one that achieves a certain minimax/maximin objective. This objective was justified on the basis of a worst-case assumption about the opponent, namely that she will play in a manner that is perfectly adversarial. Superficially, this assumption may have seemed very safe and conservative. After all, if a strategy is optimal against a perfectly adversarial opponent, then will it not also be optimal against every opponent?

As we will see in this chapter, this is emphatically not the case. The problem with the minimax/maximin objective is that it does not allow for the possibility that the opponent might make a mistake. As a result, even if a strategy is optimal in a minimax/maximin sense, it may not exploit a *failure* by the opponent to choose a best response to that strategy, and therefore may receive a payoff that is much worse than could have been achieved by a different strategy.

It may seem that, in order to exploit mistakes by the opponent, we will have to abandon our worst-case assumption. In other words, it may seem that planning for a perfectly adversarial opponent, and planning for an opponent that might make a mistake, are mutually exclusive approaches, and we will have to choose one or the other.

Fortunately, there is a way to have the best of both worlds, in a certain sense. In many games, one of the players will have more than one optimal strategy. In such cases, some have argued that the player should choose the strategy from *among her optimal strategies* that takes maximum advantage of mistakes by the opponent. A strategy with this property is called a *lexicographic optimal* strategy, and our primary objective in this chapter will be to develop algorithms that can compute these kinds of strategies.

In Section 3.1 we revisit the games we described in Chapter 2 to further motivate the need for strategies that take advantage of an opponent's mistakes. In Section 3.2 we give a formal definition of a lexicographic optimal strategy, which will make precise what we mean by a "mistake" by the opponent, and also what it means to take "maximum" advantage of those mistakes.

Our algorithms for computing lexicographic optimal strategies are based on the MW algorithm, and in Section 3.4 we give intuition for why the MW algorithm might be expected to converge to this type of strategy. Unfortunately, as we will see in Section 3.5, this intuition is not entirely sound; the MW algorithm will fail to converge to a lexicographic optimal strategy for an infinitely large class of games. We take two approaches to fixing this problem. In Section 3.6, we show that, for game matrices that satisfy a certain technical condition, the MW algorithm does indeed converge to a lexicographic optimal strategy. Introducing this condition has the somewhat unexpected side benefit of also guaranteeing that the MW algorithm will converge to a pure strategy. This feature will be useful in our application of MW to reinforcement learning in Chapter 5.

In Section 3.7, we explain that our technical condition on game matrices can be seen as somewhat restrictive. To circumvent this, we describe a set of conditions, which, if satisfied by *any* algorithm, guarantee that the algorithm can be used to efficiently compute a lexicographic optimal strategy for any game matrix. We show

that the MW algorithm "nearly" satisfies these sufficient conditions, which leads to a proposal for a minor modification of the MW algorithm.

Compared to previous approaches to computing lexicographic optimal strategies, the main advantage of our MW-based approach is that it can be feasibly applied to games in which one player has an extremely large number of strategies. The games we study and solve in Chapter 5 will have this property.

3.1 Motivation

In order to argue for the importance of strategies that take advantage of an opponent's mistakes, let us re-examine some of the games we introduced in Chapter 2.

3.1.1 Rock-Paper-Scissors

Recall the two-player zero-sum game "Rock-Paper-Scissors" from Section 2.1, and consider a variant defined by the matrix in Figure 3.1, which we will call "Rock-Paper-Scissors-Bomb". In this variant, the row player is given an additional move, called Bomb, which defeats every other move. In "Rock-Paper-Scissors", the optimal strategy for each player was the uniform distribution on all three moves, and the expected outcome was a draw. "Rock-Paper-Scissors-Bomb" is much less balanced: The minimax strategy for the row player is the pure strategy concentrated on Bomb, and *every* strategy for the column player is a maximin strategy, because no matter what choice she makes, the row player can guarantee a win by choosing Bomb.

But what if the row player makes a mistake, and fails to choose Bomb? If the column player wants to take advantage of this possibility, then she should, as before, choose the uniform distribution on all three moves. This strategy is not worse than any other, but has an expected outcome of a draw in case the row player fails to choose a best response.

	Column player		
Row player	Rock	Paper	Scissors
Rock	1/2	1	0
Paper	0	1/2	1
Scissors	1	0	1/2
Bomb	0	0	0

Figure 3.1:
Game matrix for Rock-Paper-Scissors-Bomb

3.1.2 Chess

Of course, "Rock-Paper-Scissors-Bomb" is an artificial game constructed specifically to highlight a problem with the minimax/maximin objective, but the problem can arise in real games as well. Recall our discussion of chess in Section 2.2, and consider the chess board position in Figure 3.2, known as the *Saavedra position*. The white player appears to be at a major disadvantage in this position; she only has a pawn, while the black player has a rook. Nonetheless, though it may seem improbable, there exists a winning strategy for the white player from this position, first discovered by Spanish priest Reverend Saavedra in 1895 [109]. Let us examine this position from the perspective of the black player. If she assumes that the white player is perfectly adversarial, she will regard her *every available move* as being consistent with an optimal strategy, since she believes that she has already lost the game. Of course, the Saavedra position is familiar mostly to chess experts. A typical white player probably will not play the rest of the game perfectly — indeed, American chess champion Frank Marshall once said "The hardest thing in chess is to win a won game" [121] — so it would be sensible for the black player to plan for this possibility, and continue to play as though the white player will fail to execute a perfect strategy.

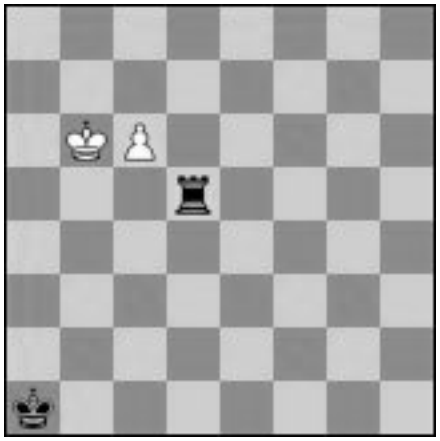

Figure 3.2: The Saavedra position. White to move and win (note that, at the start of the game, the white pieces are at the bottom of the board). This position was long believed to be a draw, until an ingenious winning strategy for white was discovered by Spanish priest Reverend Saavedra in 1895. This version of the position is due to Laskar [67].

3.1.3 Classification Game

Finally, we return to our example from machine learning. Recall the classification game from Section 2.3. In that game, the column player's choice of strategy $\mathbf{q}$ specifies a hypothesis $H_{\mathbf{q}}$, which in turn determines a margin for each example in the training set. The row player chooses a strategy $\mathbf{p}$ that specifies a distribution on the examples, and the game's payoff is the average margin of the examples with respect to $\mathbf{p}$. Therefore a best response to a column strategy by the row player is to choose a distribution $\mathbf{p}$ concentrated on the examples with the minimum margin; consequently (as we stated in Section 2.3), the column player should choose $\mathbf{q}$ so that the minimum margin of $H_{\mathbf{q}}$ on any example is as large as possible.

But how should the column player distinguish between two hypotheses $H_{\mathbf{q}_1}$ and $H_{\mathbf{q}_2}$ that induce the same minimum margin on the training set? Intuitively, choosing the hypothesis that induces larger margins on the *other* examples in the training set seems like a good idea. This approach is also supported by theoretical results that bound generalization error in terms of the entire margin distribution, and not just

minimum margin [117, 118, 35, 34]. Note that this approach amounts to planning for the possibility of a "mistake" by the row player in the classification game, specifically the possibility that she will fail to choose an example distribution concentrated on the examples with the minimum margin.

Interestingly, increasing the margins of all the examples in the training set seems to roughly correspond to what the AdaBoost classification algorithm [31], which is based on the MW algorithm, has been observed to do in practice. For example, Reyzin and Schapire [107] compare AdaBoost to arc-gv [14], another algorithm that is designed to find a hypothesis that maximizes the minimum margin on the training set. They observe that, unlike arc-gv, AdaBoost tends to increase the margins of *all* the examples in the training set (see their Figure 7). The analysis of the MW algorithm given in this chapter provides a novel explanation for this behavior.

3.2 Definition of Lexicographic Optimality

Recall that the minimax theorem [137] states

$$\min_{\mathbf{p}\in\mathbb{P}} \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{pMq} = \max_{\mathbf{q}\in\mathbb{Q}} \min_{\mathbf{p}\in\mathbb{P}} \mathbf{pMq} \triangleq v^* \tag{3.1}$$

where v^* is called the *value* of the game. As a reminder: A row strategy $\mathbf{p}^*$ that realizes the minimum in (3.1) is called a minimax strategy, while a column strategy $\mathbf{q}^*$ that realizes the maximum in (3.1) is called a maximin strategy.

Much of our discussion will apply equally well to either the row or column player, so here we focus on the column player's perspective. By examining (3.1), we see that a maximin strategy is optimal for the column player if the row player always chooses a best response, i.e., a row strategy that realizes $\min_{\mathbf{p}} \mathbf{pMq}$ for the strategy $\mathbf{q}$ that the column player chooses. As we discussed in Section 3.1, assuming that the row player will always choose a best response can have disadvantages.

How can we refine the notion of a maximin strategy to account for the possibility that the row player will make a mistake? Observe that a best response to $\mathbf{q}$ must be a row distribution that is concentrated on the smallest rows of the vector $\mathbf{Mq}$, and a mistake by the row player amounts to choosing a distribution that places any weight on any other row. Thus a column strategy $\mathbf{q}^*$ realizes the maximum in (3.1) if and only if the minimum value among the rows of the vector $\mathbf{Mq}^*$ is made as large possible (with value equal to v^*). If there is more than one such column strategy, we can take advantage of mistakes by the row player by choosing a $\mathbf{q}^*$ so that the minimum value among the *other* rows of $\mathbf{Mq}^*$ is also made as large as possible.

Extending this idea further leads to a "lexicographic" objective for the column player, which we now proceed to define formally.

3.2.1 Formal Definition

Define the *lexicographic order* $\leq_{\text{lex}}$ on $\mathbb{R}^n$ as follows: For any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq_{\text{lex}} \mathbf{y}$ if and only if $\mathbf{x} = \mathbf{y}$ or there exists a $i \in \{1, \ldots, n\}$ such that $x(j) = y(j)$ for $j < i$ and $x(i) < y(i)$. It is easily seen that $\leq_{\text{lex}}$ is a total order on $\mathbb{R}^n$. We define $\mathbf{x} <_{\text{lex}} \mathbf{y}$ to mean $\mathbf{x} \leq_{\text{lex}} \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$.

For any function $\mathbf{F} : \mathcal{X} \to \mathbb{R}^n$, define $\text{lexmax}_{x \in \mathcal{X}} \mathbf{F}(x)$ to be the maximum element in $\{\mathbf{F}(x) : x \in \mathcal{X}\}$ with respect to $\leq_{\text{lex}}$, assuming this maximum exists.

For any $\mathbf{x} \in \mathbb{R}^n$, let $\theta_i(\mathbf{x}) \in \mathbb{R}$ be the i^{th} smallest value among the components of $\mathbf{x}$. Also let $\boldsymbol{\theta}(\mathbf{x}) = (\theta_1(\mathbf{x}), \ldots, \theta_n(\mathbf{x}))$ be the values of the components of $\mathbf{x}$ sorted in nondecreasing order.

Instead of choosing a strategy that realizes the maximum in (3.1), we would like the column player to choose a *lexicographic maximin strategy* $\mathbf{q}^{\text{lex}\,*}$ that realizes

$$\mathbf{v}^* \triangleq \text{lexmax}_{\mathbf{q} \in \mathbb{Q}} \boldsymbol{\theta}(\mathbf{Mq}). \tag{3.2}$$

By definition $v^*(1) \leq \cdots \leq v^*(n)$. It is easily seen that $v^*(1) = v^*$. Therefore every lexicographic maximin strategy is also a maximin strategy. That is, every $\mathbf{q}^{\text{lex}*}$ which realizes (3.2) is also a $\mathbf{q}^*$ which realizes the maximum in (3.1). However, an advantage of $\mathbf{q}^{\text{lex}*}$ is that if the row player fails to choose a best-response strategy, then $\mathbf{q}^{\text{lex}*}$ is a maximin strategy that takes maximum advantage of this failure, in a certain sense. Basically, $\mathbf{q}^{\text{lex}*}$ is a column strategy that maximizes the smallest row of $\mathbf{Mq}$, and among all such strategies it maximizes the *second* smallest row of $\mathbf{Mq}$, and then the *third* smallest row of $\mathbf{Mq}$, and so on.

Of course, one can define a *lexicographic minimax strategy* for the row player in an analogous way, but in this chapter we will study games exclusively from the column player's perspective. A strategy that is either lexicographic minimax or lexicographic maximin is called a *lexicographic optimal strategy.*

Dresher [22] was the first to describe lexicographic optimal strategies, but did not give an explicit name to the concept; they have been termed "Dresher-optimal" strategies by most other authors (e.g. van Damme [133]). We have adopted the label "lexicographic optimal" from Ogryczak and Sliwinski [91], who defined the concept in the context of multicriteria optimization. Also, while Dresher [22]'s definition is equivalent to ours, it has a considerably different form. We discuss the relationship between the two definitions in Section 3.3.3.

3.3 Properties of the Lexicographic Optimum

In this section, we characterize properties of lexicographic maximin strategies that will be useful in the rest of the chapter.

3.3.1 Existence of a Lexicographic Optimal Strategy

The definition of $\mathbf{v}^*$ in (3.2) may prompt the following concern: Can we be certain that the set $\{\boldsymbol{\theta}(\mathbf{Mq}) : \mathbf{q} \in \mathbb{Q}\}$ contains a maximum element with respect to the lexicographic order $\leq_{\text{lex}}$? If not, then obviously $\mathbf{v}^*$ is not well-defined.

One might be tempted to prove the existence of $\mathbf{v}^*$ as a simple consequence of compactness and total orders. In other words, we might like to prove something like the following: If $E \subseteq \mathbb{R}^n$ is compact, then E contains a maximum with respect to *any* total order on $\mathbb{R}^n$. For the case $n = 1$ and the usual order $\leq$, this is certainly true, since it is equivalent to stating that if $E \subseteq \mathbb{R}$ is compact then $\sup E \in E$. Unfortunately, this fact does not extend to arbitrary total orders, as the following simple counterexample illustrates.

Claim 3.1. *There exists a nonempty compact set $E \subseteq \mathbb{R}$, and a total order $\leq_1$ on $\mathbb{R}$, such that E does not contain a maximum element with respect to $\leq_1$.*

Proof. Let $E = [0, 1]$. Clearly E is compact. Define a total order $\leq_1$ on $\mathbb{R}$ that agrees with the usual order $\leq$ except that $\leq_1$ makes 1 the least element in $\mathbb{R}$. More formally: For all $x, y \in \mathbb{R}$, if $x = 1$ then $x \leq_1 y$; if $y = 1$ then $y \leq_1 x$; otherwise $x \leq_1 y \Leftrightarrow x \leq y$. Clearly E does not contain a maximum with respect to $\leq_1$. □

The problem encountered in the counterexample is that, with respect to the order $\leq_1$, the number 1 is less than every other number, even the numbers that it is "close to". So, in order to prove the existence of $\mathbf{v}^*$, we must verify that this pathology does not apply to the lexicographic order $\leq_{\text{lex}}$.

The existence of $\mathbf{v}^*$ in (3.2) is implied by the following theorem.

Theorem 3.2. *Let $\mathcal{X}$ be a compact metric space. For any continuous function $\mathbf{F} : \mathcal{X} \to \mathbb{R}^n$ the set $\{\boldsymbol{\theta}(\mathbf{F}(x)) : x \in \mathcal{X}\}$ contains a maximum with respect to $\leq_{\text{lex}}$.*

Because $\mathbf{Mq}$ is a continuous function of $\mathbf{q}$, and because the set of all column

strategies $\mathbb{Q}$ is a compact metric space, Theorem 3.2 implies that $\mathbf{v}^*$ in (3.2) exists, and thus that a column strategy $\mathbf{q}^{\text{lex}\,*}$ that realizes the lexicographic maximum exists.

To prove Theorem 3.2, we will need the next two lemmas.

Lemma 3.3. *Every nonempty compact subset of $\mathbb{R}^n$ contains a maximum element with respect to $\leq_{\text{lex}}$.*

Proof. Let $E \in \mathbb{R}^n$ be a nonempty compact set. Define $E_0 = E$, and define $X_1 \dots X_n$ and $E_1 \dots E_n$ as follows: Let $X_i = \{x(i) : \mathbf{x} \in E_{i-1}\}$ and let $E_i = \{\mathbf{x} \in E_{i-1} : x(i) = \sup X_i\}$.

We claim that $E_0 \dots E_n$ are all nonempty and compact. By definition this holds for E_0. Now suppose for induction that E_{i-1} is nonempty and compact for some $i \geq 1$. Then clearly X_i is nonempty and compact (because $E_{i-1} \subseteq \mathbb{R}^n$), which implies that $\sup X_i \in X_i$, which implies that E_i is nonempty. Also observe that $E_i = E_{i-1} \cap (\mathbb{R}^{i-1} \times \{\sup X_i\} \times \mathbb{R}^{n-i})$, which is clearly compact (because compactness is preserved under both Cartesian product and intersection). The claims holds by induction.

Let $\mathbf{x}^* \in E_n$. We claim that $\mathbf{x} \leq_{\text{lex}} \mathbf{x}^*$ for all $\mathbf{x} \in E$, which proves the lemma. For any fixed $\mathbf{x} \in E$, if $\mathbf{x} = \mathbf{x}^*$, then $\mathbf{x} \leq_{\text{lex}} \mathbf{x}^*$ trivially. Otherwise let i be the smallest integer such that $x(i) \neq x^*(i)$. Since $\mathbf{x}^* \in E_n \subseteq E_i$, we have $x^*(j) = \sup X_j$ for $j \in \{1, \dots, i\}$. Therefore $x(j) = \sup X_j$ for $j \in \{1, \dots, i-1\}$, which implies that $\mathbf{x} \in E_{i-1}$. We conclude that $x(i) < x^*(i)$, and thus $\mathbf{x} \leq_{\text{lex}} \mathbf{x}^*$. □

Lemma 3.4. *The sorting function $\boldsymbol{\theta} : \mathbb{R}^n \to \mathbb{R}^n$ is continuous.*

Proof. Fix $\mathbf{x} \in \mathbb{R}^n$ and $\epsilon > 0$. Let $\delta = \min(\epsilon, \frac{\delta'}{2})$, where δ' is any value that satisfies $0 < \delta' \leq |x(i) - x(j)|$ for all $i, j \in \{1, \dots, n\}$ such that $x(i) \neq x(j)$. Let $E = \{\mathbf{y} : \|\mathbf{y} - \mathbf{x}\|_\infty < \delta\}$.

We claim that $\|\boldsymbol{\theta}(\mathbf{x}) - \boldsymbol{\theta}(\mathbf{y})\|_\infty < \epsilon$ for all $\mathbf{y} \in E$, which proves the lemma. Observe that for any $\mathbf{y} \in E$ and $i, j \in \{1, \dots, n\}$, if $x(i) < x(j)$ then $y(i) < y(j)$.

This is because $\delta \leq \frac{\delta'}{2}$. Fix $\mathbf{y} \in E$, and let $\pi_{\mathbf{y}}$ be a permutation of the indices of $\mathbf{y}$ which leads to $\boldsymbol{\theta}(\mathbf{y})$, and let $\pi_{\mathbf{x}}$ be the same for $\mathbf{x}$. Clearly we can choose $\pi_{\mathbf{y}}$ and $\pi_{\mathbf{x}}$ so that $\pi_{\mathbf{y}} = \pi_{\mathbf{x}}$. Since $\|\mathbf{x} - \mathbf{y}\|_\infty < \delta$ we have $\|\boldsymbol{\theta}(\mathbf{x}) - \boldsymbol{\theta}(\mathbf{y})\|_\infty < \delta \leq \epsilon$. And this holds for all $\mathbf{y} \in E$ because we fixed $\mathbf{y}$ arbitrarily. □

Proof of Theorem 3.2. Since $\mathbf{F}(x)$ is continuous, Lemma 3.4 implies that $\boldsymbol{\theta}(\mathbf{F}(x))$ is continuous. Since $\mathcal{X}$ is compact, we have that $E = \{\boldsymbol{\theta}(\mathbf{F}(x)) : x \in \mathcal{X}\}$ is compact, because the image of a continuous function from a compact metric space into a metric space is compact [110]. Thus Lemma 3.3 implies that E contains a maximum element with respect to $\leq_{\text{lex}}$. □

3.3.2 Equivalence of All Lexicographic Optimal Strategies

While there may be more than one column strategy $\mathbf{q}^{\text{lex}*}$ that realizes (3.2), all such strategies are equivalent in a certain sense, as established by the next theorem.

Theorem 3.5. *For any pair of column strategies* $\mathbf{q}_1^{\text{lex}*}$ *and* $\mathbf{q}_2^{\text{lex}*}$ *if*

$$\mathbf{v}^* = \boldsymbol{\theta}(\mathbf{M}\mathbf{q}_1^{\text{lex}*}) = \boldsymbol{\theta}(\mathbf{M}\mathbf{q}_2^{\text{lex}*})$$

then $\mathbf{M}\mathbf{q}_1^{\text{lex}*} = \mathbf{M}\mathbf{q}_2^{\text{lex}*}$.

Proof. Assume without loss of generality that the rows of $\mathbf{M}$ are ordered so that $\boldsymbol{\theta}(\mathbf{M}\mathbf{q}_1^{\text{lex}*}) = \mathbf{M}\mathbf{q}_1^{\text{lex}*} = \mathbf{v}^*$. We will prove that $\mathbf{M}\mathbf{q}_2^{\text{lex}*} = \mathbf{v}^*$.

For contradiction, assume $\mathbf{M}\mathbf{q}_2^{\text{lex}*} \neq \mathbf{v}^*$, and let i be the smallest row index such that $M(i, \mathbf{q}_2^{\text{lex}*}) \neq v^*(i)$. This implies that $M(j, \mathbf{q}_2^{\text{lex}*}) \geq v^*(i)$ for $j \geq i$ (because otherwise there would exist a row $j \leq i$ such that $\theta_j(\mathbf{M}\mathbf{q}_2^{\text{lex}*}) < v^*(j)$, which cannot happen since $\boldsymbol{\theta}(\mathbf{M}\mathbf{q}_2^{\text{lex}*}) = \mathbf{v}^*$). Therefore, by our choice of i, we have $M(i, \mathbf{q}_2^{\text{lex}*}) > v^*(i)$.

Now let $\tilde{\mathbf{q}} = \frac{1}{2}\mathbf{q}_1^{\text{lex}*} + \frac{1}{2}\mathbf{q}_2^{\text{lex}*}$, so we have

$$\mathbf{M}\tilde{\mathbf{q}} = \frac{1}{2}\mathbf{M}\mathbf{q}_1^{\text{lex}*} + \frac{1}{2}\mathbf{M}\mathbf{q}_2^{\text{lex}*} = \frac{1}{2}\mathbf{v}^* + \frac{1}{2}\mathbf{M}\mathbf{q}_2^{\text{lex}*}$$

Let $i' \geq i$ be the largest row index such that $v^*(i) = v^*(i')$. In other words, $v^*(i) = v^*(i+1) = \cdots = v^*(i'-1) = v^*(i')$.

We have shown the following:

- $M(j, \tilde{\mathbf{q}}) = v^*(j)$ for $1 \leq j < i$ (because $M(j, \mathbf{q}_2^{\text{lex}*}) = v^*(j)$ for $1 \leq j < i$).
- $M(i, \tilde{\mathbf{q}}) > v^*(i)$ (because $M(i, \mathbf{q}_2^{\text{lex}*}) > v^*(i)$).
- $M(j, \tilde{\mathbf{q}}) \geq v^*(i)$ for $i < j \leq i'$ (because $M(j, \mathbf{q}_2^{\text{lex}*}) \geq v^*(i)$ and $v^*(j) \geq v^*(i)$ for $j \geq i$).
- $M(j, \tilde{\mathbf{q}}) > v^*(i)$ for $j > i'$ (because $M(j, \mathbf{q}_2^{\text{lex}*}) \geq v^*(i)$ for $j \geq i$ and $v^*(j) > v^*(i)$ for $j > i'$).

By examining this list, one can see that it implies $\boldsymbol{\theta}(\mathbf{M}\tilde{\mathbf{q}}) >_{\text{lex}} \mathbf{v}^*$, which contradicts the definition of $\mathbf{v}^*$.

□

Theorem 3.5 is quite convenient. Since $\mathbf{M}\mathbf{q}^{\text{lex}*}$ is the same for every lexicographic maximin strategy $\mathbf{q}^{\text{lex}*}$, we may assume without loss of generality that the rows of $\mathbf{M}$ are ordered so that $\boldsymbol{\theta}(\mathbf{M}\mathbf{q}^{\text{lex}*}) = \mathbf{M}\mathbf{q}^{\text{lex}*} = \mathbf{v}^*$ for every such strategy. We make this assumption throughout the rest of this chapter.

3.3.3 Alternate Characterization of Lexicographic Optimum

We now give an alternate characterization of $\mathbf{v}^*$, which will allow us to relate the concept of lexicographic optimality to earlier work. But first, let us recall our discussion of restricted games in Section 2.5. In a restricted game, the row and column player

must choose their strategies from subsets $\mathcal{P} \subseteq \mathbb{P}$ and $\mathcal{Q} \subseteq \mathbb{Q}$, respectively. To aid our alternate characterization of $\mathbf{v}^*$, we will examine two particular families of such subsets, one for each player. Recalling our notation from Section 2.5, let us define $\mathbb{P}_{-i} \triangleq \mathbb{P}_{\{i+1,\ldots,n\}}$ for each row $1 \leq i \leq n-1$. In other words,

$$\mathbb{P}_{-i} \triangleq \{\mathbf{p} \in \mathbb{P} : p(j) = 0 \text{ for } 1 \leq j \leq i\}$$

So $\mathbb{P}_{-i}$ is the set of row strategies concentrated on *all but* the first i rows of $\mathbf{M}$. Now, for each row $1 \leq i \leq n$, we also define

$$\mathbb{Q}_i \triangleq \{\mathbf{q} \in \mathbb{Q} : M(i, \mathbf{q}) = v^*(i) \text{ for } 1 \leq j \leq i\}$$

In other words, if $\mathbf{q} \in \mathbb{Q}_i$, then the first i rows of $\mathbf{Mq}$ must match the first i rows of $\mathbf{Mq}^{\text{lex}*}$.

For notational convenience, we define $\mathbb{P}_{-0} \triangleq \mathbb{P}$ and $\mathbb{Q}_0 \triangleq \mathbb{Q}$. With these definitions in place, we can state the following theorem.

Theorem 3.6. *For each* $0 \leq i \leq n-1$

$$v^*(i+1) = \max_{\mathbf{q} \in \mathbb{Q}_i} \min_{\mathbf{p} \in \mathbb{P}_{-i}} \mathbf{pMq} = \min_{\mathbf{p} \in \mathbb{P}_{-i}} \max_{\mathbf{q} \in \mathbb{Q}_i} \mathbf{pMq}$$

The significance of Theorem 3.6 is twofold. Firstly, the statement of the theorem is the *definition* of $\mathbf{v}^*$ in earlier work on lexicographic optimality ([22, 133]), so this theorem shows that our definition and the original one are equivalent. Secondly, the theorem suggests an approach for computing $\mathbf{q}^{\text{lex}*}$. Since each component of $\mathbf{v}^*$ is the value of a different zero-sum game, finding a lexicographic maximin strategy in a game reduces to finding an ordinary maximin strategy in each of several smaller games. This is precisely the approach taken by Miltersen and Sorensen [79], whose algorithm we describe in the next section.

Proof of Theorem 3.6. Because $\mathbf{q}^{\text{lex}*} \in \mathbb{Q}_i$ we have

$$\max_{\mathbf{q}\in\mathbb{Q}_i} \min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{pMq} \geq \min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{pMq}^{\text{lex}*} = v^*(i+1) \tag{3.3}$$

Now let $\mathbf{q}' = \arg\max_{\mathbf{q}\in\mathbb{Q}_i} \min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{pMq}$. We have

$$v^*(i+1) \geq \theta_{i+1}(\mathbf{Mq}') \geq \min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{pMq}' = \max_{\mathbf{q}\in\mathbb{Q}_i} \min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{pMq} \tag{3.4}$$

The first inequality follows because $\mathbf{v}^* \geq_{\text{lex}*} \boldsymbol{\theta}(\mathbf{Mq}')$ and $\mathbf{q}' \in \mathbb{Q}_i$. The second inequality holds for any column strategy $\mathbf{q}$. Together (3.3) and (3.4) imply that $v^*(i+1) = \max_{\mathbf{q}\in\mathbb{Q}_i} \min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{pMq}$.

The fact that

$$\max_{\mathbf{q}\in\mathbb{Q}_i} \min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{pMq} = \min_{\mathbf{p}\in\mathbb{P}_{-i}} \max_{\mathbf{q}\in\mathbb{Q}_i} \mathbf{pMq} \tag{3.5}$$

follows from our discussion of restricted games in Section 2.5. Since $\mathbb{P}_{-i} \subseteq \mathbb{P}$ and $\mathbb{Q}_i \subseteq \mathbb{Q}$ are both closed and convex, Theorems 2.9 and 2.10 together imply (3.5).

□

3.4 Approaches to Computing Lexicographic Optimal Strategies

In Section 2.4.1, we showed how the optimal strategies in a two-player zero-sum game can be computed by solving a certain linear program. A similar approach due to Miltersen and Sorensen [79] can be used to compute lexicographic optimal strategies. However, instead of solving a single linear program, their procedure solves a *sequence* of them, where each linear program in the sequence solves a different game. The ith game in the sequence corresponds to the characterization of $v^*(i)$ given in Theorem 3.6.

Here is a sketch of the procedure described by Miltersen and Sorensen [79] for computing $\mathbf{q}^{\text{lex}*}$. The first linear program in the sequence outputs a description of the sets $\mathbb{Q}_1$ and $\mathbb{P}_{-1}$ (defined in Section 3.3.3). The next linear program uses $\mathbb{Q}_1$ and $\mathbb{P}_{-1}$ to output $\mathbb{Q}_2$ and $\mathbb{P}_{-2}$. The procedure continues in this manner until $\mathbb{Q}_n$ is output, which only contains $\mathbf{q}^{\text{lex}*}$ by definition. As we discussed in Section 2.4.1, a linear program can be solved in time that is polynomial in the size of the program, so the running time of this entire procedure is polynomial in the size of the game matrix $\mathbf{M}$.

In this rest of this chapter, we will take a markedly different approach to computing a lexicographic optimal strategy than Miltersen and Sorensen [79]. Recall that in Section 2.4.3 we showed how, in a repeated game, if the row player chooses strategies according the MW algorithm, and the column player always chooses a best response, then the average of the column strategies converges to a maximin strategy. In this chapter, we will prove that, under very similar conditions, those column strategies will converge to a *lexicographic* maximin strategy. This convergence is not automatic, however; in Section 3.5 we prove that, without additional assumptions or modifications, the MW algorithm may fail to converge to a strategy with the desired property. In Section 3.6 we give a technical condition that ensures convergence to a lexicographic maximin strategy. A side benefit of this condition is that it allows us to prove convergence of the *last* column strategy, not just the average column strategy. Since each column strategy generated during the repeated game is a pure strategy, this allows us to prove convergence to a pure lexicographic maximin strategy, a very useful guarantee, as we will see in Chapter 5 when we apply it to a reinforcement learning problem.

In Section 3.7, we explain that our technical condition on game matrices is somewhat restrictive. To avoid this, we describe a set of conditions on no-regret algorithms which suffice for lexicographic convergence. In particular, we prove that if the row

player chooses her strategies according to an algorithm that satisfies these conditions, and the column player always chooses a best response, then the average column strategy converges to a lexicographic optimal strategy, for any game matrix. We show that the MW algorithm "nearly" satisfies these sufficient conditions, which leads to a proposal for a minor modification of the MW algorithm.

The main advantage of using the MW algorithm instead of solving a sequence of linear programs is that, unlike the LP-based approach, the running time of the MW algorithm has *no explicit dependence* on the number of columns of the game matrix $\mathbf{M}$, provided it has access to a best-response oracle for the column player. As we will see in Chapter 5, these advantages are particularly appropriate for the games we are interested in solving.

Why should we expect that the MW algorithm will converge to a lexicographic maximin strategy? Some intuition can be gained from the following discussion. Assume that the initial distribution $\mathbf{p}_1$ is set to be uniform. Then in each round t of the MW algorithm, the weight assigned by row distribution $\mathbf{p}_t$ to row i is

$$p_t(i) \propto \beta^{\sum_{t'=1}^{t-1} M(i,\mathbf{q}_{t'})} = \beta^{(t-1)M(i,\overline{\mathbf{q}}_{t-1})} \tag{3.6}$$

where $\overline{\mathbf{q}}_{t-1} = \frac{1}{t-1}\sum_{t'=1}^{t-1} \mathbf{q}_{t'}$ is the average of the column strategies up to round $t-1$. Because $\beta < 1$, this means that $\mathbf{p}_t$ places the most weight on the "hardest" row up to round $t-1$, i.e., the row $i^{\min}$ such that $M(i^{\min}, \overline{\mathbf{q}}_{t-1})$ is smallest. Therefore, if $\mathbf{q}_t$ is chosen to be a best response to $\mathbf{p}_t$, it will tend to be chosen so that $M(i^{\min}, \mathbf{q}_t)$ is large. This is precisely the reason why, once the MW algorithm has completed all T rounds, the smallest row of $\mathbf{M}\overline{\mathbf{q}}_T$ has been made nearly as large as possible, and thus $\overline{\mathbf{q}}_T$ is an approximate maximin strategy. Essentially, the distribution $\mathbf{p}_t$ forces the column player to focus on the hardest rows of $\mathbf{M}$.

But (3.6) also implies that $\mathbf{p}_t$ assigns the *second* most weight to the second hardest

row of $\mathbf{M}$, and that the *third* most weight to the third hardest row of $\mathbf{M}$, etc. At least superficially, this pattern suggests that the MW algorithm might be computing a lexicographic maximin strategy. This turns out to be *almost* true. In the next section we show exactly where this reasoning breaks down.

3.5 Divergence of MW Algorithm

When the MW algorithm is used to compute a maximin strategy for a game matrix $\mathbf{M}$, it may fail to converge to a lexicographic maximin strategy. The high-level intuition for why this occurs is that the algorithm may choose $\mathbf{q}_t$'s that focus *too much* on the "hard" rows of the game matrix $\mathbf{M}$, at the cost of ignoring the other rows.

Theorem 3.7. *For all $a \in (0, \frac{1}{2}]$ and $\epsilon \in (0, \frac{a}{4})$ there exists a 3×4 matrix $\mathbf{M}_{a,\epsilon}$ for which the following holds: Suppose the repeated game is played on $\mathbf{M}_{a,\epsilon}$ for T rounds. Suppose the row player chooses strategies $\mathbf{p}_1, \ldots, \mathbf{p}_T$ according to the MW algorithm, with parameters $\alpha_1, \ldots, \alpha_3 > 0$ and $\beta \in (0, 1)$. Suppose the column player chooses a best response $\mathbf{q}_t = \arg\max_{\mathbf{q} \in \mathbb{Q}} \mathbf{p}_t \mathbf{M}_{a,\epsilon} \mathbf{q}$ in each round t. Then*

$$\liminf_{T \to \infty} \left\| \boldsymbol{\theta}(\mathbf{M}_{a,\epsilon} \mathbf{q}_T) - \mathbf{v}^* \right\|_\infty \geq a - 2\epsilon$$

Note that Theorem 3.7 immediately implies that

$$\liminf_{T \to \infty} \left\| \boldsymbol{\theta}(\mathbf{M}_{a,\epsilon} \overline{\mathbf{q}}_T) - \mathbf{v}^* \right\|_\infty \geq a - 2\epsilon$$

since increasing the number of rounds in the repeated game (while keeping the other parameters fixed) does not affect the behavior of the MW algorithm during the earlier rounds.

Proof of Theorem 3.7. The matrix $\mathbf{M}_{a,\epsilon}$ is

$$\mathbf{M}_{a,\epsilon} = \begin{bmatrix} a+\epsilon & a-\epsilon & a+2\epsilon & a-2\epsilon \\ a-\epsilon & a+\epsilon & a-2\epsilon & a+2\epsilon \\ 2a & 2a & a & a \end{bmatrix}$$

Note that our choices for a and ϵ guarantee that the entries of $\mathbf{M}_{a,\epsilon}$ lie in the interval $[0, 1]$.

Let $\mathbf{q}^j$ be the pure column strategy concentrated on column j. It is straightforward to verify that $\mathbf{q}^{\text{lex}\,*} = \frac{1}{2}\mathbf{q}^1 + \frac{1}{2}\mathbf{q}^2$, and thus

$$\mathbf{v}^* = \begin{bmatrix} a \\ a \\ 2a \end{bmatrix}$$

Also, it is clear that for all $\mathbf{q} \in \mathbb{Q}$, we have $M_{a,\epsilon}(i, \mathbf{q}) \leq a + 2\epsilon$ for $i \in \{1, 2\}$. So it suffices to prove that $M_{a,\epsilon}(3, \mathbf{q}_T) \leq a + 2\epsilon$ for all sufficiently large T. In fact, we will prove $M_{a,\epsilon}(3, \mathbf{q}_T) \leq a + \epsilon$.

The basic idea of the proof is that for all rounds t — except for finitely many — $p_t(3)$ is much smaller than the difference $|p_t(1) - p_t(2)|$. Therefore, when the column player chooses a best response to $\mathbf{p}_t$, it is better for her to choose $\mathbf{q}^3$ over $\mathbf{q}^1$, and $\mathbf{q}^4$ over $\mathbf{q}^2$.

Recall that $p_t(i) \propto w_t(i)$, where $w_t(i)$ is the weight on row i in round t of the MW algorithm. For each round t, let $i_t^-, i_t^+ \in \{1, 2\}$ be such that row i_t^+ has at least the weight of row i_t^- in round t, i.e., $w_t(i_t^-) \leq w_t(i_t^+)$. Also, let $r_t = \frac{w_t(i_t^-)}{w_t(i_t^+)}$ be the smaller ratio of the weights on rows 1 and 2 in round t.

We make the following observation, which will be useful throughout the remainder of the proof: For each round t, if $r_t < 1$ (i.e., there is an imbalance of weights on rows 1 and 2), then $M_{a,\epsilon}(i_t^-, \mathbf{q}_t) < M_{a,\epsilon}(i_t^+, \mathbf{q}_t)$ (because $\mathbf{q}_t$ is a best response to $\mathbf{p}_t$), and

therefore by the definition of the MW algorithm

$$w_t(i_t^-)\beta^{a-\epsilon} \leq w_{t+1}(i_t^-) \leq w_t(i_t^-)\beta^{a-2\epsilon} \tag{3.7}$$

$$w_t(i_t^+)\beta^{a+2\epsilon} \leq w_{t+1}(i_t^+) \leq w_t(i_t^+)\beta^{a+\epsilon} \tag{3.8}$$

We claim that $r_{t_0} \geq \beta^{4\epsilon}$ for some round $t_0 \leq \frac{\log r_1}{2\epsilon \log \beta}$. Consider a round t such that $r_t < \beta^{4\epsilon}$. By (3.7) and (3.8) we have

$$\frac{w_{t+1}(i_t^-)}{w_{t+1}(i_t^+)} \leq \frac{w_t(i_t^-)\beta^{a-2\epsilon}}{w_t(i_t^+)\beta^{a+2\epsilon}} = r_t\beta^{-4\epsilon} < \beta^{4\epsilon}\beta^{-4\epsilon} = 1$$

which implies that $i_{t+1}^+ = i_t^+$ and $i_{t+1}^- = i_t^-$. We can now conclude, again using (3.7) and (3.8), that

$$r_{t+1} = \frac{w_{t+1}(i_{t+1}^-)}{w_{t+1}(i_{t+1}^+)} = \frac{w_{t+1}(i_t^-)}{w_{t+1}(i_t^+)} \geq \frac{w_t(i_t^-)\beta^{a-\epsilon}}{w_t(i_t^+)\beta^{a+\epsilon}} = r_t\beta^{-2\epsilon}$$

Note that $\beta^{-2\epsilon} > 1$. Thus, by repeatedly applying this recurrence, we have that $r_t < \beta^{4\epsilon}$ for at most the first $\frac{\log r_1}{2\epsilon \log \beta} - 1$ rounds, which proves the claim.

We claim that if $r_t \geq \beta^{4\epsilon}$ then $r_{t+1} \geq \beta^{4\epsilon}$. We divide into two cases: (i) $i_{t+1}^+ = i_t^+$ and $i_{t+1}^- = i_t^-$; (ii) $i_{t+1}^+ = i_t^-$ and $i_{t+1}^- = i_t^+$. For case (i), we can simply apply the recurrence from above to get

$$r_{t+1} \geq r_t\beta^{-2\epsilon} > r_t \geq \beta^{4\epsilon}$$

because $\beta^{-2\epsilon} > 1$. For case (ii), we have by (3.7) and (3.8)

$$r_{t+1} = \frac{w_{t+1}(i_{t+1}^-)}{w_{t+1}(i_{t+1}^+)} = \frac{w_{t+1}(i_t^+)}{w_{t+1}(i_t^-)} \geq \frac{w_t(i_t^+)\beta^{a+2\epsilon}}{w_t(i_t^-)\beta^{a-2\epsilon}} = \frac{1}{r_t}\frac{\beta^{a+2\epsilon}}{\beta^{a-2\epsilon}} \geq \frac{\beta^{a+2\epsilon}}{\beta^{a-2\epsilon}} = \beta^{4\epsilon}$$

because $r_t \leq 1$.

To recap, we have shown that $r_t \geq \beta^{4\epsilon}$ for all $t \geq t_0$. We will now use this fact to

argue that the number of rounds $t \geq t_0$ such that $M_{a,\epsilon}(3, \mathbf{q}_t) > a + \epsilon$ is bounded.

Choose any $t \geq t_0$ such that $M_{a,\epsilon}(3, \mathbf{q}_t) > a + \epsilon$. Note that $\mathbf{q}_t$, a distribution on the columns of $\mathbf{M}_{a,\epsilon}$, must have support that intersects the support of either $\mathbf{q}^1$ or $\mathbf{q}^2$. Since $\mathbf{q}_t$ is a best response to $\mathbf{p}_t$, one of the following inequalities must hold:

$$\mathbf{p}_t \mathbf{M}_{a,\epsilon} \mathbf{q}^1 \geq \mathbf{p}_t \mathbf{M}_{a,\epsilon} \mathbf{q}^3, \text{ or}$$
$$\mathbf{p}_t \mathbf{M}_{a,\epsilon} \mathbf{q}^2 \geq \mathbf{p}_t \mathbf{M}_{a,\epsilon} \mathbf{q}^4$$

Whichever is the case, multiplying both sides of the appropriate inequality by $\sum_i w_t(i)$ (which is the normalization constant for $\mathbf{p}_t$) gives

$$(a+\epsilon)w_t(i_t^+) + (a-\epsilon)w_t(i_t^-) + 2aw_t(3) \geq (a+2\epsilon)w_t(i_t^+) + (a-2\epsilon)w_t(i_t^-) + aw_t(3)$$

Simplifying, we get

$$aw_t(3) \geq \epsilon \left(w_t(i_t^+) - w_t(i_t^-) \right) \tag{3.9}$$

Now choose $\delta, \delta' \geq 0$ so that $w_t(i_t^+) = \beta^{ta-\delta}$ and $w_t(i_t^-) = \beta^{ta+\delta'}$. By the definition of the MW algorithm we know that $w_t(i) = \beta^{\sum_{t'=1}^{t-1} M_{a,\epsilon}(i, \mathbf{q}_{t'})}$ for all rows i. So by the symmetry of the matrix $\mathbf{M}_{a,\epsilon}$ we have $\delta = \delta'$. Since $t \geq t_0$, then because $r_t = \frac{w_t(i_t^-)}{w_t(i_t^+)} \geq \beta^{4\epsilon}$ we have $\delta \leq 2\epsilon$. Thus we can conclude

$$w_t(i_t^+) - w_t(i_t^-) \geq \beta^{ta} \left(\beta^{-2\epsilon} - \beta^{2\epsilon} \right) \tag{3.10}$$

Now let τ_0 be the number of rounds t' such that $t_0 \leq t' < t$ and $M_{a,\epsilon}(3, \mathbf{q}_{t'}) > a + \epsilon$. We have

$$w_t(3) \leq \beta^{ta + \tau_0 \epsilon} \tag{3.11}$$

Combining (3.9), (3.10), and (3.11) and dividing both sides by β^{ta} yields

$$a\beta^{\tau_0\epsilon} \geq \epsilon\left(\beta^{-2\epsilon} - \beta^{2\epsilon}\right)$$

and solving for τ_0 we get $\tau_0 \leq \frac{\epsilon(\beta^{-2\epsilon}-\beta^{2\epsilon})-\log a}{\epsilon \log \beta}$.

Since $t \geq t_0$ was an arbitrarily chosen round for which $M_{a,\epsilon}(3, \mathbf{q}_t) > a+\epsilon$, we have that there are at most $t_0+\tau_0+1$ rounds such that $M_{a,\epsilon}(3, \mathbf{q}_t) > a+\epsilon$. Now observe that for a fixed choice of parameters $\alpha_1, \ldots, \alpha_3$ and β, and for any $T' \leq T$, the behavior of the MW algorithm during the first T' rounds is independent of the total number of rounds T (under the mild assumption that the tie-breaking procedure when choosing $\mathbf{q}_t = \arg\max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}_t \mathbf{M}_{a,\epsilon}\mathbf{q}$ is independent of T). Thus, for all sufficiently large T, we have $M_{a,\epsilon}(3, \mathbf{q}_T) \leq a + \epsilon$. □

3.6 Convergence of MW Under a Strictness Condition

In this section, we prove that a certain assumption about the game matrix $\mathbf{M}$ guarantees convergence of the MW algorithm to a lexicographic maximin strategy. Recall that by definition $v^*(1) \leq \ldots \leq v^*(n)$. Our key assumption is that all these inequalities are *strict*.

Assumption 3.8 (Strictness)**.** *Define* $\delta_{\mathbf{M}} \triangleq \min_{i\neq j} |v^*(i) - v^*(j)|$. *We have* $\delta_{\mathbf{M}} > 0$.

Intuitively, we might expect that a small value for $\delta_{\mathbf{M}}$ will cause problems for convergence, since when $\delta_{\mathbf{M}}$ is small Assumption 3.8 is close to being violated. Indeed, the magnitude of $\delta_{\mathbf{M}}$ will constrain the values of the parameter β for which our convergence theorem holds. In general, we must have $\beta \geq 2/3$, and when $\delta_{\mathbf{M}}$ is small, β must lie in an even smaller interval. Our convergence theorem, which we state next, makes this precise.

Theorem 3.9. *Suppose Assumption 3.8 holds. Suppose the repeated game is played on a matrix* $\mathbf{M}$ *for* T *rounds. Suppose the row player chooses strategies* $\mathbf{p}_1, \ldots, \mathbf{p}_T$ *according to the MW algorithm, with parameters* $\alpha_1, \ldots, \alpha_n > 0$ *and* $\beta \geq 1/(1 + \delta_{\mathbf{M}}/2)$. *Suppose the column player chooses a pure strategy* $\mathbf{q}_t$ *satisfying* $\mathbf{q}_t = \arg\max_{\mathbf{q} \in \mathbb{Q}} \mathbf{p}_t \mathbf{M} \mathbf{q}$ *in each round* t. *Then*

$$\lim_{T \to \infty} \mathbf{M} \mathbf{q}_T = \mathbf{v}^*$$

Note that Theorem 3.9 is a result about the *last* column strategy output by the MW algorithm, not the average column strategy $\overline{\mathbf{q}}_T$ as in Theorem 2.8 (although of course Theorem 3.9 immediately implies that $\lim_{T \to \infty} \mathbf{M} \overline{\mathbf{q}}_T = \mathbf{v}^*$). Also note that requiring that $\mathbf{q}_t$ be a pure strategy is without loss of generality, since there is always a best response with this property.

3.6.1 Sketch of Proof

It is possible to provide a general understanding for why Theorem 3.9 is true without delving into the details.

Recall that in each round t of the repeated game, the weight assigned by row distribution $\mathbf{p}_t$ to row i is

$$p_t(i) \propto \alpha_i \beta^{\sum_{t'=1}^{t-1} M(i, \mathbf{q}_{t'})} = \alpha_i \beta^{(t-1) M(i, \overline{\mathbf{q}}_{t-1})} \tag{3.12}$$

where $\overline{\mathbf{q}}_{t-1} = \frac{1}{t-1} \sum_{t'=1}^{t-1} \mathbf{q}_{t'}$ is the average of the column strategies up to round $t-1$.

One consequence of Assumption 3.8 is that, as the repeated game proceeds, a "gap" appears between $M(1, \overline{\mathbf{q}}_t)$ and $M(i, \overline{\mathbf{q}}_T)$ for all rows $i > 1$. More precisely, there exists an $\epsilon > 0$ such that $M(1, \overline{\mathbf{q}}_t) < M(i, \overline{\mathbf{q}}_t) - \epsilon$ for all rows $i > 1$ and all sufficiently large t. In any round t where this is true, we have for all $i > 1$

$$\frac{p_t(1)}{p_t(i)} \geq \frac{\alpha_1}{\alpha_i} \beta^{-\epsilon(t-1)} \tag{3.13}$$

which follows from (3.12). Note that this lower bound is an exponentially large quantity in t. Thus, in the later rounds of the repeated game, the distribution $\mathbf{p}_t$ places an overwhelming fraction of its weight on row 1.

Another consequence of Assumption 3.8 is the following:

$$\text{If } \mathbf{q} \in \mathbb{Q}_{i-1} \text{ then } M(i, \mathbf{q}) \leq v^*(i) \tag{3.14}$$

For the case $i = 1$, we see that *no* column strategy $\mathbf{q} \in \mathbb{Q}$ can assign a higher value than $v^*(1)$ to the first row of $\mathbf{Mq}$.

Combining (3.13) and (3.14), we can show that there must be a round t_1 such that for *all* rounds $t \geq t_1$ we have $M(1, \mathbf{q}_t) = v^*(1)$. In other words, when $t \geq t_1$, row 1 is assigned so much weight by $\mathbf{p}_t$ that a best response to $\mathbf{p}_t$ must be a column strategy $\mathbf{q}_t$ that maximizes $M(1, \mathbf{q}_t)$ — and this maximum is equal to $v^*(1)$. Note that this implies that $\mathbf{q}_t \in \mathbb{Q}_1$. And clearly, among all the column strategies in $\mathbb{Q}_1$, the column player will choose $\mathbf{q}_t$ so that it is a best response to the row distribution $\mathbf{p}_{-1,t} \in \mathbb{P}_{-1}$, which we define below to be $\mathbf{p}_t$ restricted to *all but* the first row.

Now, if we examine the repeated game from round t_1 onward, we see that it corresponds exactly to an instance of a restricted repeated game on the matrix $\mathbf{M}$, where the row player's strategies are restricted to $\mathcal{P} = \mathbb{P}_{-1}$ and the column player's strategies are restricted to $\mathcal{Q} = \mathbb{Q}_1$. So we can simply repeat the argument from above for this restricted repeated game. There is one complication: The parameters $\alpha_1, \ldots, \alpha_n$ in the first round of the restricted repeated game — i.e. round t_1 — are not set uniformly across the last $n-1$ rows of $\mathbf{M}$, but rather are set so they match the row distribution $\mathbf{p}_{t_1}$. But this turns out not to offer much difficulty in the analysis. So we can repeat the argument from above and conclude that there is a round t_2 such that for *all* rounds $t \geq t_2$ we have $M(2, \mathbf{q}_t) = v^*(2)$. Continuing in this manner yields the theorem.

3.6.2 Complete Proof

Before proceeding with the proof, let us establish some additional shorthand. Recalling our notation for restricted row strategies from Section 2.5, let

$$\mathbf{p}_{-i,t} \triangleq \mathbf{p}_{\{i+1,\dots,n\},t}$$
$$Z_{-i,t} \triangleq Z_{\{i+1,\dots,n\},t}$$

In other words, the row strategy $\mathbf{p}_{-i,t} \in \mathbb{P}_{-i}$ is $\mathbf{p}_t$ restricted to *all but* the first i rows of $\mathbf{M}$, and $Z_{-i,t}$ is its normalization constant. For convenience, define $\mathbf{p}_{-0,t} \triangleq \mathbf{p}_t$ and $Z_{-0,t} \triangleq Z_t$.

Our first two lemmas provide useful characterizations of the set $\mathbb{Q}_{i-1}$. Their proofs use Assumption 3.8, and neither lemma is true if that assumption does not hold.

Lemma 3.10. *If* $\mathbf{q} \in \mathbb{Q}_{i-1}$ *then* $M(i, \mathbf{q}) \leq v^*(i)$

Proof. Assume for contradiction that $M(i, \mathbf{q}) > v^*(i)$. Consider the column strategy

$$\mathbf{q}_\lambda = \lambda \mathbf{q} + (1 - \lambda)\mathbf{q}^{\text{lex}\,*}$$

for some $\lambda \in [0, 1]$. Recall that $\mathbf{M}\mathbf{q}^{\text{lex}\,*} = \mathbf{v}^*$. Since both $\mathbf{q}, \mathbf{q}^{\text{lex}\,*} \in \mathbb{Q}_{i-1}$, we have $M(j, \mathbf{q}_\lambda) = v^*(j)$ for all $j \leq i - 1$. Also, if $\lambda > 0$, then $M(i, \mathbf{q}_\lambda) > v^*(i)$ by our assumption. Moreover, as $\lambda \to 0$, the value of $M(j, \mathbf{q}_\lambda)$ becomes arbitrarily close to $v^*(j)$ for all j. And since $v^*(j) > v^*(i)$ for all $j > i$ (by Assumption 3.8), we can choose a value for λ such that $\boldsymbol{\theta}(\mathbf{M}\mathbf{q}_\lambda) >_{\text{lex}} \boldsymbol{\theta}(\mathbf{M}\mathbf{q}^{\text{lex}\,*})$, which is a contradiction. □

Lemma 3.11. *Let* $\mathbf{p}_t$ *be a row strategy generated by the MW algorithm, and recall that* $\mathbf{q}_t = \arg\max_{\mathbf{q} \in \mathbb{Q}} \mathbf{p}_t \mathbf{M}\mathbf{q}$. *If* $\mathbf{q}_t \in \mathbb{Q}_{i-1}$, *then*

$$\mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_t = \max_{\mathbf{q} \in \mathbb{Q}_i} \mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}$$

Proof. Let $\mathbf{q}_i^* = \arg\max_{\mathbf{q}\in\mathbb{Q}_i} \mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}$. Clearly $\mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_t \leq \mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_i^*$. We also have

$$\begin{aligned}
\mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_t &= \frac{1}{Z_{-i,t}} \sum_{j>i} p_t(j) M(j, \mathbf{q}_t) \\
&= \frac{1}{Z_{-i,t}} \left(\sum_j p_t(j) M(j, \mathbf{q}_t) - \sum_{j\leq i} p_t(j) M(j, \mathbf{q}_t) \right) \\
&\geq \frac{1}{Z_{-i,t}} \left(\sum_j p_t(j) M(j, \mathbf{q}_i^*) - \sum_{j\leq i} p_t(j) M(j, \mathbf{q}_t) \right) \qquad (3.15) \\
&\geq \frac{1}{Z_{-i,t}} \left(\sum_j p_t(j) M(j, \mathbf{q}_i^*) - \sum_{j\leq i} p_t(j) M(j, \mathbf{q}_i^*) \right) \qquad (3.16) \\
&= \frac{1}{Z_{-i,t}} \sum_{j>i} p_t(j) M(j, \mathbf{q}_i^*) \\
&= \mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_i^*
\end{aligned}$$

In (3.15) we used $\mathbf{q}_t = \arg\max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}_t\mathbf{M}\mathbf{q} = \arg\max_{\mathbf{q}\in\mathbb{Q}} \sum_j p_t(j) M(j, \mathbf{q})$.

To prove (3.16), note that since $\mathbf{q}_i^* \in \mathbb{Q}_i$, we have

$$M(j, \mathbf{q}_i^*) = v^*(j) \text{ for } j \leq i \qquad (3.17)$$

And since $\mathbf{q}_t \in \mathbb{Q}_{i-1}$ by assumption, we can conclude by Lemma 3.10 that

$$M(j, \mathbf{q}_t) = v^*(j) \text{ for } j \leq i-1, \text{ and } M(i, \mathbf{q}_t) \leq v^*(i) \qquad (3.18)$$

Thus (3.17) and (3.18) allow us to conclude (3.16). □

The next two lemmas are the main tools used to prove Theorem 3.9.

Lemma 3.12. *Suppose the MW algorithm is run for T rounds. If there exists $1 \leq i \leq n-1$ such that $\mathbf{q}_t \in \mathbb{Q}_{i-1}$ for all $1 \leq t \leq T$, then*

$$\min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T \geq v^*(i+1) - \Delta_{T,\mathbf{p}_1,\beta}$$

Before proceeding with its proof, it is instructive to pause here and contemplate the implications of Lemma 3.12. The conditions of Lemma 3.12 certainly hold for $i = 1$, since we always have $\mathbf{q}_t \in \mathbb{Q}_0$, so let us consider this case. Corollary 2.4 and Lemma 3.12 together imply that, if $\Delta_{T,\mathbf{p}_1,\beta}$ is small, then after T rounds the smallest element of $\mathbf{M}\overline{\mathbf{q}}_T$ is not much less than $v^*(1)$, and *also* that the *second* smallest element of $\mathbf{M}\overline{\mathbf{q}}_T$ is not much less than $v^*(2)$. This is the property that will allow us to prove convergence to a lexicographic optimum.

Proof of Lemma 3.12. Let $\overline{\mathbf{p}}_{-i,T} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_{-i,t}$ be the average of the restricted row strategies, and note that $\overline{\mathbf{p}}_{-i,T} \in \mathbb{P}_{-i}$. Consider this chain of equalities/inequalities:

$$
\begin{aligned}
v^*(i+1) &= \min_{\mathbf{p}\in\mathbb{P}_{-i}} \max_{\mathbf{q}\in\mathbb{Q}_i} \mathbf{pMq} && (3.19)\\
&\leq \max_{\mathbf{q}\in\mathbb{Q}_i} \overline{\mathbf{p}}_{-i,T}\mathbf{Mq} && (3.20)\\
&= \max_{\mathbf{q}\in\mathbb{Q}_i} \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_{-i,t}\mathbf{Mq} && (3.21)\\
&\leq \frac{1}{T}\sum_{t=1}^{T} \max_{\mathbf{q}\in\mathbb{Q}_i} \mathbf{p}_{-i,t}\mathbf{Mq} \\
&= \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_{-i,t}\mathbf{Mq}_t && (3.22)\\
&\leq \min_{\mathbf{p}\in\mathbb{P}_{-i}} \frac{1}{T}\sum_{t=1}^{T} \mathbf{pMq}_t + \Delta_{T,\mathbf{p}_1,\beta} && (3.23)\\
&= \min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{pM}\overline{\mathbf{q}}_T + \Delta_{T,\mathbf{p}_1,\beta}
\end{aligned}
$$

In (3.19) we used Theorem 3.6. In (3.20) and (3.21) we used the definition of $\overline{\mathbf{p}}_{-i,T}$. In (3.21) we used Lemma 3.11. In (3.23) we used Theorem 2.10. □

Our final lemma requires one additional definition. Consider an arbitrary matrix entry $M(i,j)$, and suppose that $M(i,j) < v_i$. Then by the finiteness of the matrix $\mathbf{M}$, there must exist a constant $\gamma > 0$ such that $v^*(i) - M(i,j) \geq \gamma$. We will let γ be

the largest constant for which this inequality holds for all choices of i and j. In other words

$$\gamma \triangleq \min_{(i,j):v^*(i)>M(i,j)} v^*(i) - M(i,j)$$

Lemma 3.13. *Suppose the MW algorithm is run for $T > \frac{2\log((n-i)/\gamma)}{\delta_{\mathbf{M}}\log(1/\beta)}$ rounds. If there exists $1 \le i \le n-1$ such that $\mathbf{q}_t \in \mathbb{Q}_{i-1}$ for all $1 \le t \le T$ and*

$$\min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T \ge v^*(i+1) - \delta_{\mathbf{M}}/2,$$

then $\mathbf{q}_T \in \mathbb{Q}_i$.

Proof. Since $\mathbf{q}_T \in \mathbb{Q}_{i-1}$ by assumption, we only need to show that $M(i,\mathbf{q}_T) = v^*(i)$. By Lemma 3.10 we have that $M(i,\mathbf{q}_T) \le v^*(i)$, so assume for contradiction that $M(i,\mathbf{q}_T) < v^*(i)$. Consider this chain of equalities/inequalities:

$$\begin{aligned}
\mathbf{p}_T\mathbf{M}\mathbf{q}_T - \mathbf{p}_T\mathbf{M}\mathbf{q}^{\text{lex}\,*} &= \sum_{j=1}^{n} p_T(j)(M(j,\mathbf{q}_T) - v^*(j)) \\
&\le -\gamma p_T(i) + \sum_{j=i+1}^{n} p_T(j) && (3.24)\\
&= p_T(i)\left(-\gamma + \sum_{j=i+1}^{n} \frac{p_T(j)}{p_T(i)}\right) \\
&= p_T(i)\left(-\gamma + \sum_{j=i+1}^{n} \frac{\beta^{TM(j,\overline{\mathbf{q}}_T)}}{\beta^{TM(i,\overline{\mathbf{q}}_T)}}\right) && (3.25)\\
&\le p_T(i)\left(-\gamma + (n-i)\beta^{T(\delta_{\mathbf{M}}/2)}\right) && (3.26)\\
&< 0 && (3.27)
\end{aligned}$$

where we used the following:

In (3.24): Since $\mathbf{q}_T \in \mathbb{Q}_{i-1}$, we have $M(j,\mathbf{q}_T) = v^*(j)$ for $1 \le j \le i-1$. And since $\mathbf{q}_T$ is a pure strategy and $M(i,\mathbf{q}_T) < v^*(i)$ by assumption, we have $M(i,\mathbf{q}_T) \le v^*(i) - \gamma$ by the definition of γ.

In (3.25): Definition of MW algorithm.

In (3.26): Since $\mathbf{q}_t \in \mathbb{Q}_{i-1}$ for $1 \leq t \leq T$ by assumption, by Lemma 3.10 we have that $M(i, \mathbf{q}_t) \leq v^*(i)$, and hence that $M(i, \overline{\mathbf{q}}_T) \leq v^*(i)$. By assumption $M(j, \overline{\mathbf{q}}_T) \geq v^*(i{+}1) - \delta_{\mathbf{M}}/2$ for all $j \geq i{+}1$. And by Assumption 3.8, we have $v^*(i{+}1) - v^*(i) \geq \delta_{\mathbf{M}}$.

In (3.27): Plugging in assumption about T.

Note that $\mathbf{p}_T \mathbf{M} \mathbf{q}_T - \mathbf{p}_T \mathbf{M} \mathbf{q}^{\text{lex}*} < 0$ contradicts the choice of $\mathbf{q}_T$. □

We are now ready to prove Theorem 3.9.

Proof of Theorem 3.9. We will just repeatedly apply Lemmas 3.12 and 3.13. We divide the rounds of the MW algorithm into phases $i = 1, \ldots, n$, with phase i beginning at round t_i. We will prove the following claim by induction: If T is sufficiently large, then there exist choices for $t_1, \ldots, t_{n+1}$ such that for all rounds $t \geq t_i$ we have $\mathbf{q}_t \in \mathbb{Q}_{i-1}$.

The claim is clearly true for $i = 1$ by letting $t_1 = 1$. Suppose it holds for all $1 \leq i' \leq i$. Consider a "shadow" instance of the MW algorithm beginning at round t_i and running for T_i rounds, where the parameters $\alpha_1, \ldots, \alpha_n$ are set to match the distribution $\mathbf{p}_{t_i}$. Observe that this "shadow" instance exactly matches the behavior of the original instance of the MW algorithm during these rounds. By Lemma 3.12, we have $\min_{\mathbf{p} \in \mathbb{P}_{-i}} \mathbf{p} \mathbf{M} \overline{\mathbf{q}}_{T_i} \geq v^*(i{+}1) - \Delta_{T_i, \mathbf{p}_{t_i}, \beta}$. By our assumptions about $\alpha_1, \ldots, \alpha_n$ and β, we have that $\Delta_{T_i, \mathbf{p}_{t_i}, \beta}$ is finite and approaches $\delta_{\mathbf{M}}/3$ from above as $T_i \to \infty$. Thus we can choose T_i large enough so that we have $\Delta_{T, \mathbf{p}_{t_i}, \beta} \leq \delta_{\mathbf{M}}/2$ for all $T \geq T_i$, and therefore by Lemma 3.13 we have $\mathbf{q}_T \in \mathbb{Q}_i$ for all $T \geq T_i$. So let $t_{i+1} = t_i + T_i$. □

3.7 Algorithm-Based Sufficient Conditions

While Assumption 3.8 allows us to prove convergence of the MW algorithm to a lexicographic optimum, it can be quite a limiting assumption, as revealed by the next theorem.

Theorem 3.14. *Suppose Assumption 3.8 holds. Then, in the game defined by the matrix* $\mathbf{M}$*, there exists a pure lexicographic maximin strategy for the column player, and a unique (and pure) minimax strategy for the row player.*

Proof. The statement of Theorem 3.9 supplies a proof of the existence of a pure lexicographic maximin strategy. Indeed, the theorem states that the last column strategy $\mathbf{q}_T$ generated during the repeated game is a lexicographic maximin strategy (if T is sufficiently large), and that the column strategy generated in each round is pure.

To show that there exists a pure minimax strategy, we must first establish a simple claim: If $\mathbf{p}^*$ and $\mathbf{q}^*$ are minimax and maximin strategies, respectively, then $\mathbf{p}^*$ is a best-response to $\mathbf{q}^*$. Indeed

$$\min_{\mathbf{p}\in\mathbb{P}} \mathbf{p}\mathbf{M}\mathbf{q}^* = v^* = \mathbf{p}^*\mathbf{M}\mathbf{q}^*$$

which proves the claim.

Let $\mathbf{q}^{\text{lex}\,*}$ be a lexicographic maximin strategy. We know that $\mathbf{q}^{\text{lex}\,*}$ is also a maximin strategy. So, by the previous claim, any minimax strategy $\mathbf{p}^*$ is a best-response to $\mathbf{q}^{\text{lex}\,*}$. But, by Assumption 3.8, the column vector $\mathbf{M}\mathbf{q}^{\text{lex}\,*}$ has its unique smallest component in row 1, and so $\mathbf{p}^*$ must be the pure row strategy concentrated on row 1. Thus $\mathbf{p}^*$ is the unique minimax strategy. □

In Section 2.2, we showed that there exists a large class of games for which both players have pure optimal strategies — namely, extensive form games of perfect information. This is a huge family of games, so Assumption 3.8 — even in light of Theorem 3.14 — does not necessarily rule out every game we might be interested in solving. Still, there are many games for which the optimal strategies for one or both players must be mixed.

We would like an algorithm that can compute a lexicographic maximin strategy

in any game, even those for which Assumption 3.8 does not hold. And we would also like to preserve the desirable properties of the MW algorithm, such as its ability to solve games in which one player has a very large number of strategies. So in this section, instead of describing conditions on the game matrix which guarantee lexicographic convergence of the MW algorithm, we will describe conditions on *any* algorithm which guarantee lexicographic convergence, for any game matrix.

To this end, let us re-examine the reason why the MW algorithm, when used to solve a game, can fail to converge to a lexicographic maximin strategy. In Section 3.5, we described an infinite family of matrices for which this failure occurs. For each matrix $\mathbf{M}_{a,\epsilon}$ in the family, we had

$$\mathbf{v}^* = \begin{bmatrix} a \\ a \\ 2a \end{bmatrix} \quad \text{and} \quad \mathbf{M}_{a,\epsilon}\mathbf{q}_T \approx \begin{bmatrix} a \\ a \\ a \end{bmatrix}$$

for all sufficiently large T, where a is a positive constant. At a high-level, there was one essential difficulty faced by the MW algorithm when it was applied to the matrix $\mathbf{M}_{a,\epsilon}$ — the value of $p_t(3)$ rapidly approached zero as t became large. As a result, when $\mathbf{q}_t$ was chosen to be a best-response to $\mathbf{p}_t$, this best-response tended to "ignore" row 3, and hence $M_{a,\epsilon}(3, \mathbf{q}_t)$ was not as large as it could be.

Since the difficulty is that $\mathbf{p}_t$ assigns too little weight to some rows, one solution might be to artificially force the weight on all rows to be above some lower bound. Indeed, as we will see, essentially *any* no-regret algorithm which satisfies this simple property can be used to compute a lexicographic maximin strategy.

Before we can state the main result of this section, we need to establish some additional shorthand. Recalling our notation for restricted row strategies from Section

2.5, let

$$\mathbb{P}_i \triangleq \mathbb{P}_{\{1,\ldots,i\}}$$

$$\mathbf{p}_{i,t} \triangleq \mathbf{p}_{\{1,\ldots,i\},t}$$

$$Z_{i,t} \triangleq Z_{\{1,\ldots,i\},t}$$

In other words, $\mathbb{P}_i$ is the set of row strategies concentrated on the first i rows of $\mathbf{M}$. The row strategy $\mathbf{p}_{i,t} \in \mathbb{P}_i$ is $\mathbf{p}_t$ restricted to the first i rows of $\mathbf{M}$, and $Z_{i,t}$ is its normalization constant.

We are now ready to state the main result of this section.

Theorem 3.15. *Suppose the repeated game is played on a matrix* $\mathbf{M}$ *for* T *rounds. Suppose the row player chooses strategies* $\mathbf{p}_1, \ldots, \mathbf{p}_T$*, and the column player chooses a strategy* $\mathbf{q}_t = \arg\max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}_t\mathbf{M}\mathbf{q}$ *in each round* t*. Suppose there exists* $i \in \{0, \ldots, n-1\}$ *and scalars* ϵ_T *and* C *such that*

(a) $\frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_{i,t}\mathbf{M}\mathbf{q}_t \leq \min_{\mathbf{p}\in\mathbb{P}_i} \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}\mathbf{M}\mathbf{q}_t + \epsilon_T$

(b) $\frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_t \leq \min_{\mathbf{p}\in\mathbb{P}_{-i}} \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}\mathbf{M}\mathbf{q}_t + \epsilon_T$

(c) $Z_{-i,t} \geq C$ *for all rounds* t

(d) $\overline{Z}_{-i,T} \leq C + \epsilon_T$

where $\overline{Z}_{-i,T} \triangleq \frac{1}{T}\sum_{t=1}^{T} Z_{-i,t}$*. Then*

$$\min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T \geq v^*(i+1) - \frac{4\epsilon_T}{C}$$

Conditions (a) and (b) of Theorem 3.15 simply require that row player's regret with respect to the first i rows, and also the last $n-i$ rows, is at most ϵ_T. Conditions (c) and (d) place upper and lower bounds on the weight assigned by $\mathbf{p}_t$ to the last $n-i$ rows.

Note that the statement of the theorem immediately implies

$$\theta_{i+1}(\mathbf{M}\overline{\mathbf{q}}_T) \geq v^*(i+1) - \frac{4\epsilon_T}{C}$$

because $\theta_{i+1}(\mathbf{Mq}) \geq \min_{\mathbf{p}\in\mathbb{P}_{-i}} \mathbf{pMq}$ for any column strategy $\mathbf{q}$.

3.7.1 Satisfiability of Conditions

Before proving Theorem 3.15, we should ask whether its conditions are reasonable. Is there an algorithm for choosing $\mathbf{p}_1, \ldots, \mathbf{p}_T$ so that the conditions of Theorem 3.15 are satisfied? The next theorem proves that the MW algorithm suffices, albeit with a very strong caveat.

Define $\tilde{\imath}$ to be the largest row index such that $v^*(\tilde{\imath}) = v^*(1)$. Therefore, $v^*(1) = \cdots = v^*(\tilde{\imath}) < v^*(\tilde{\imath}+1) \leq \cdots \leq v^*(n)$. Or, put differently, $\tilde{\imath}$ is the index of the first "breakpoint" in the sequence of values $v^*(1), \ldots, v^*(n)$.

Recall our definition of $\delta_{\mathbf{M}}$ (in Assumption 3.8) as the minimum difference between $v^*(i)$ and $v^*(j)$ for any distinct pair i and j. Let us define

$$\delta^v_{\mathbf{M}} \triangleq \min_{v^*(i)\neq v^*(j)} |v^*(i) - v^*(j)|$$

as the minimum difference between any distinct pair $v^*(i)$ and $v^*(j)$.

Theorem 3.16. *Suppose the repeated game is played on a matrix* $\mathbf{M}$ *for* T *rounds. Suppose the row player chooses strategies* $\mathbf{p}_1, \ldots, \mathbf{p}_T$ *according to the MW algorithm, and the column player chooses a strategy* $\mathbf{q}_t = \arg\max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}_t\mathbf{Mq}$ *in each round* t*. Then the conditions of Theorem 3.15 are satisfied for all* $i \in \{\tilde{\imath} \ldots n-1\}$*, with* $\epsilon_T = \Delta_{T,\mathbf{p}_1,\beta}/\delta^v_{\mathbf{M}}$ *and* $C = 0$*.*

Proof. By Theorem 2.10 we immediately have

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}_{i,t}\mathbf{M}\mathbf{q}_t \leq \min_{\mathbf{p}\in\mathbb{P}_i}\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}\mathbf{M}\mathbf{q}_t + \Delta_{T,\mathbf{p}_1,\beta}$$

and

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_t \leq \frac{1}{T}\min_{\mathbf{p}\in\mathbb{P}_{-i}}\sum_{t=1}^{T}\mathbf{p}\mathbf{M}\mathbf{q}_t + \Delta_{T,\mathbf{p}_1,\beta}$$

which proves conditions (a) and (b). Clearly $Z_{-i,t} \geq 0$ for all rounds t, which proves condition (c).

We claim that $\overline{Z}_{-i,T} \leq \Delta_{T,\mathbf{p}_1,\beta}/\delta^v_{\mathbf{M}}$, which proves condition (d) and completes the proof of the theorem. Recall from Theorem 2.8 that

$$\max_{\mathbf{q}\in\mathbb{Q}}\overline{\mathbf{p}}_T\mathbf{M}\mathbf{q} \leq v^*(1) + \Delta_{T,\mathbf{p}_1,\beta} \tag{3.28}$$

However

$$\max_{\mathbf{q}\in\mathbb{Q}}\overline{\mathbf{p}}_T\mathbf{M}\mathbf{q} \geq \overline{\mathbf{p}}_T\mathbf{M}\mathbf{q}^{\text{lex}\,*} \geq \overline{Z}_{i,T}v^*(1) + \overline{Z}_{-i,T}(v^*(1) + \delta^v_{\mathbf{M}}) \tag{3.29}$$

where $\overline{Z}_{i,T} \triangleq \frac{1}{T}\sum_{t=1}^{T} Z_{i,t}$, and the last inequality follows from that fact that $M(j, \mathbf{q}^{\text{lex}\,*}) \geq v^*(1) + \delta^v_{\mathbf{M}}$ for all $j \in \{\tilde{\imath}, \ldots, n\}$. Combining (3.28) and (3.29), and noting that $\overline{Z}_{i,T} + \overline{Z}_{-i,T} = 1$, proves the claim. □

It may appear at first glance that Theorem 3.16 implies that the column strategy $\overline{\mathbf{q}}_T$ output by the MW algorithm converges to a lexicographic maximin strategy. Indeed, for all $i \in \{0 \ldots \tilde{\imath} - 1\}$, we have

$$\theta_{i+1}(\mathbf{M}\overline{\mathbf{q}}_T) \geq v^*(i+1) - \Delta_{T,\mathbf{p}_1,\beta} \tag{3.30}$$

by the original analysis of the MW algorithm in Theorem 2.8, and for all $i \in \{\tilde{\imath} \ldots n-1\}$ we have

$$\theta_{i+1}(\mathbf{M}\overline{\mathbf{q}}_T) \geq v^*(i+1) - \frac{4}{C\delta^v_{\mathbf{M}}}\Delta_{T,\mathbf{p}_1,\beta} \tag{3.31}$$

by Theorems 3.15 and 3.16. Since $\Delta_{T,\mathbf{p}_1,\beta}$ can be made arbitrarily close to zero for sufficiently large T, (3.30) and (3.31) together imply that $\overline{\mathbf{q}}_T$ is nearly a lexicographically optimal maximin strategy, if C is a positive constant.

The difficulty, of course, is that Theorem 3.16 only tells us that MW satisfies the conditions of Theorem 3.15 for $C = 0$, and so the bound in (3.31) is vacuous. Nonetheless, Theorem 3.16 suggests that the MW algorithm is "almost" suitable for computing lexicographic maximin strategies, and perhaps only needs to be modified slightly. We explore this possibility in Section 3.7.4.

3.7.2 Proof of Convergence

We are now ready to prove the main result of this section.

Proof of Theorem 3.15. By the definition of $\mathbf{q}^{\text{lex}*}$ we have

$$\mathbf{p}_t \mathbf{M} \mathbf{q}^{\text{lex}*} = \sum_j p_t(j) v^*(j) \geq Z_{i,t} v^*(1) + Z_{-i,t} v^*(i+1) \tag{3.32}$$

where the inequality uses the fact that $v^*(1) \leq \cdots \leq v^*(n)$. We also have

$$\mathbf{p}_t \mathbf{M} \mathbf{q}_t = Z_{i,t} \mathbf{p}_{i,t} \mathbf{M} \mathbf{q}_t + Z_{-i,t} \mathbf{p}_{-i,t} \mathbf{M} \mathbf{q}_t \tag{3.33}$$

Since $\mathbf{p}_t \mathbf{M} \mathbf{q}_t \geq \mathbf{p}_t \mathbf{M} \mathbf{q}^{\text{lex}*}$ by the choice of $\mathbf{q}_t$, we can combine (3.32) and (3.33) and rearrange terms to find

$$Z_{i,t} \left[\mathbf{p}_{i,t} \mathbf{M} \mathbf{q}_t - v^*(1) \right] + Z_{-i,t} \mathbf{p}_{-i,t} \mathbf{M} \mathbf{q}_t \geq Z_{-i,t} v^*(i+1)$$

which will be convenient to re-write as

$$\left[Z_{i,t} - (1-C) + (1-C) \right] \left[\mathbf{p}_{i,t} \mathbf{M} \mathbf{q}_t - v^*(1) \right] + \left[Z_{-i,t} - C + C \right] \mathbf{p}_{-i,t} \mathbf{M} \mathbf{q}_t \geq Z_{-i,t} v^*(i+1)$$

By condition (c) we have $Z_{-i,t} \geq C$, and since $Z_{i,t}+Z_{-i,t} = 1$, condition (c) also implies that $1 - C \geq Z_{i,t}$. Applying these inequalities, as well as the obvious inequalities $\mathbf{p}_{i,t}\mathbf{M}\mathbf{q}_t - v^*(1) \geq -1$ and $\mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_t \leq 1$, and averaging both sides over all T rounds yields

$$(1-C) - \overline{Z}_{i,T} + (1-C)\frac{1}{T}\sum_{t=1}^{T}\left[\mathbf{p}_{i,t}\mathbf{M}\mathbf{q}_t - v^*(1)\right] + \overline{Z}_{-i,T} - C + C\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_t \geq \overline{Z}_{-i,T}v^*(i+1)$$

Applying conditions (a), (c) and (d) to this expression and simplifying gives us

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_t \geq v^*(i+1) - \frac{3\epsilon_T}{C} \tag{3.34}$$

By condition (b) we have

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}_{-i,t}\mathbf{M}\mathbf{q}_t \leq \min_{\mathbf{p}\in\mathbb{P}_{-i}}\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}\mathbf{M}\mathbf{q}_t + \epsilon_T \tag{3.35}$$

Combining (3.34) and (3.35) proves the theorem. □

3.7.3 Convergence of Rows

Theorem 3.15 shows that, if a particular row index i satisfies a certain set of conditions, then the $i+1$st best component of the vector $\mathbf{M}\overline{\mathbf{q}}_T$ is (approximately) lower-bounded by $v^*(i+1)$. When the conditions of Theorem 3.15 are satisfied for *all* $j \in \{0, \ldots, i\}$, we can further characterize $\mathbf{M}\overline{\mathbf{q}}_T$. Stated informally, the next theorem shows that not only will the top $i+1$ components of $\mathbf{M}\overline{\mathbf{q}}_T$ be lower-bounded by the top $i+1$ components of $\mathbf{v}^*$ (respectively), but additionally the first $i+1$ rows of $\mathbf{M}\overline{\mathbf{q}}_T$ will have *converged* to the first $i+1$ rows of $\mathbf{v}^*$ (respectively).

Theorem 3.17. *Suppose that for all* $j \in \{0, \ldots, i\}$

$$\liminf_{T\to\infty} \min_{\mathbf{p}\in\mathbb{P}_{-j}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T \geq v^*(j+1)$$

Then for all $j \in \{0, \ldots, i\}$

$$\lim_{T\to\infty} M(j+1, \overline{\mathbf{q}}_T) = v^*(j+1)$$

Proof. Choose any $i' \in \{0, \ldots, i\}$, and suppose that for all $j \in \{0, \ldots, i'-1\}$ we have

$$\lim_{T\to\infty} M(j+1, \overline{\mathbf{q}}_T) = v^*(j+1) \tag{3.36}$$

and

$$\liminf_{T\to\infty} \min_{\mathbf{p}\in\mathbb{P}_{-i'}} \mathbf{p}\mathbf{M}\overline{\mathbf{q}}_T \geq v^*(i'+1) \tag{3.37}$$

Now consider the following claim:

$$\lim_{T\to\infty} M(i'+1, \overline{\mathbf{q}}_T) = v^*(i'+1) \tag{3.38}$$

In the rest of the proof, we will show that (3.36) and (3.37) imply (3.38). It is easy to check that this proves the theorem.

Fix $i' \in \{0, \ldots, i\}$ and $\epsilon > 0$. Let

$$\mathbb{Q}_\epsilon = \{\mathbf{q} : M(i'+1, \mathbf{q}) \geq v^*(i'+1) + \epsilon\}$$

Since (3.37) implies that $M(i'+1, \overline{\mathbf{q}}_T) \geq v^*(i'+1) - \epsilon$ for all sufficiently large T, it suffices to show that $\overline{\mathbf{q}}_T \notin \mathbb{Q}_\epsilon$ for all sufficiently large T.

Let $\tilde{\mathbf{q}} \in \mathbb{Q}_{i'} \cap \mathbb{Q}_\epsilon$. We claim that $\min_{\mathbf{p}\in\mathbb{P}_{-i'}} \mathbf{p}\mathbf{M}\tilde{\mathbf{q}} < v^*(i'+1)$. Suppose for contradiction that $\min_{\mathbf{p}\in\mathbb{P}_{-i'}} \mathbf{p}\mathbf{M}\tilde{\mathbf{q}} \geq v^*(i'+1)$. Thus we have the following:

- $M(j, \tilde{\mathbf{q}}) = v^*(j)$ for $1 \le j \le i'$ (because $\tilde{\mathbf{q}} \in \mathbb{Q}_i$).
- $M(i'+1, \tilde{\mathbf{q}}) > v^*(i'+1)$ (because $\tilde{\mathbf{q}} \in \mathbb{Q}_\epsilon$).
- $M(j, \tilde{\mathbf{q}}) \ge v^*(i'+1)$ for $i'+1 < j \le n$ (because $\min_{\mathbf{p} \in \mathbb{P}_{-i'}} \mathbf{pM}\tilde{\mathbf{q}} \ge v^*(i'+1)$).

By examining this list, one can see that it implies $\boldsymbol{\theta}(\mathbf{M}\tilde{\mathbf{q}}) >_{\text{lex}} \mathbf{v}^*$, which contradicts the definition of $\mathbf{v}^*$.

Now, for each $\mathbf{q}$, let $E(\mathbf{q})$ be the following set of at most $i'+1$ elements:

$$E(\mathbf{q}) = \{|M(j, \mathbf{q}) - v^*(j)| : 1 \le j \le i'\} \cup \left\{ \max\left(0, v^*(i'+1) - \min_{\mathbf{p} \in \mathbb{P}_{-i'}} \mathbf{pMq}\right) \right\}$$

and let $E = \{\min E(\mathbf{q}) : \mathbf{q} \in \mathbb{Q}_\epsilon\}$. If there exists $\mathbf{q} \in \mathbb{Q}_\epsilon$ such that $\min E(\mathbf{q}) = 0$, then $\mathbf{q} \in \mathbb{Q}_i$ and $\min_{\mathbf{p} \in \mathbb{P}_{-i}} \mathbf{pMq} \ge v^*(i+1)$. But this contradicts our claim above. Hence $x > 0$ for all $x \in E$.

It is easy to see that E is compact, and therefore $\inf E \in E$. Let $\epsilon' = \inf E > 0$. By the conditions of the lemma, for all sufficiently large T we have $\min E(\overline{\mathbf{q}}_T) < \epsilon'$, and thus $\overline{\mathbf{q}}_T \notin \mathbb{Q}_\epsilon$. □

3.7.4 The MW(λ) Algorithm

The conditions of Theorem 3.15 essentially require two things: that the row strategies have the no-regret property for the first i rows and the last $n-i$ rows, and that the weights on the last $n-i$ rows are lower- and upper-bounded. As we saw in Theorem 3.16, while the MW algorithm has the first of these properties, its weights are only trivially lower-bounded by zero, rendering the guarantees of Theorem 3.15 vacuous.

How might we ensure that the weights assigned to each row by the MW algorithm are lower-bounded by a constant? Here is one simple approach: replace $\mathbf{p}_t$ with $\mathbf{p}_t^\lambda$, where $\mathbf{p}_t^\lambda = (1-\lambda)\mathbf{p}_t + \lambda\mathbf{u}$, and $\mathbf{u}$ is the uniform distribution on all the rows, and $\lambda \in [0,1]$. Note that $\mathbf{p}_t^\lambda(i) \ge \frac{\lambda}{n}$ for all rows i. We call this the MW(λ) algorithm, and

we will apply it in Chapter 5 to a reinforcement learning problem.

We will now prove that the MW(λ) algorithm converges to the lexicographic maximin column strategy for the matrices $\mathbf{M}_{a,\epsilon}$ from the proof of Theorem 3.7, for which we have already shown that the MW algorithm provably diverges.

Theorem 3.18. *Let $\mathbf{M}_{a,\epsilon}$ be as in the proof of Theorem 3.7, for any $a \in (0, \frac{1}{2}]$ and $\epsilon \in (0, \frac{a}{4})$. Suppose the repeated game is played on $\mathbf{M}_{a,\epsilon}$ for T rounds. Suppose the row player chooses strategies $\mathbf{p}_1^\lambda, \ldots, \mathbf{p}_T^\lambda$ according to the MW(λ) algorithm, with parameters $\alpha_1, \ldots, \alpha_3$ and β tuned as in Lemma 2.5 so that $\Delta_{T,\mathbf{p}_1,\beta} = \Delta_{T,n}$. Suppose the column player chooses a best response $\mathbf{q}_t = \arg\max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}_t^\lambda \mathbf{M}_{a,\epsilon}\mathbf{q}$ in each round t. Then*

$$\theta_i(\mathbf{M}_{a,\epsilon}\overline{\mathbf{q}}_T) \geq v^*(i) - \max\left\{\Delta_{T,n} + \lambda, \frac{12\Delta_{T,n}}{\lambda\delta_{\mathbf{M}}^v}\right\}$$

for all rows i.

Proof. We claim that $\theta_1(\mathbf{M}_{a,\epsilon}\overline{\mathbf{q}}_T) \geq v^*(1) - (\Delta_{T,n} + \lambda)$. Since $v^*(1) = v^*(2)$, this immediately implies that $\theta_2(\mathbf{M}_{a,\epsilon}) \geq v^*(2) - (\Delta_{T,n} + \lambda)$.

Consider the repeated game in which the row player chooses $\mathbf{p}_1, \ldots, \mathbf{p}_T$ and the column player chooses $\mathbf{q}_1, \ldots, \mathbf{q}_T$. We know that the row player's average regret in this repeated game is at most $\Delta_{T,n}$. And since $\mathbf{p}_t^\lambda \mathbf{M}_{a,\epsilon}\mathbf{q}_t = \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}^\lambda \mathbf{M}_{a,\epsilon}\mathbf{q}$ by the choice of $\mathbf{q}_t$, and $\mathbf{p}^\lambda = (1-\lambda)\mathbf{p}_t + \lambda\mathbf{u}$, we have that

$$\mathbf{p}_t \mathbf{M}_{a,\epsilon}\mathbf{q}_t + \lambda \geq \max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}^\lambda \mathbf{M}_{a,\epsilon}\mathbf{q}$$

So the column player's average regret in this repeated game is at most λ. A direct application of Theorem 2.8 proves the claim.

To prove that $\theta_3(\mathbf{M}_{a,\epsilon}\overline{\mathbf{q}}_T) \geq v^*(3) - \frac{12\Delta_{T,n}}{\lambda\delta_{\mathbf{M}}^v}$, we will simply verify that the conditions of Theorem 3.15 hold with $\epsilon_T = \frac{\Delta_{T,n}}{\delta_{\mathbf{M}}^v}$ and $C = \frac{\lambda}{3}$.

To prove condition (a), we inspect the matrix $\mathbf{M}_{a,\epsilon}$ and observe that by the choice of $\mathbf{q}_t$ we have $M_{a,\epsilon}(1, \mathbf{q}_t) \geq M_{a,\epsilon}(2, \mathbf{q}_t)$ if and only if $p_t^\lambda(1) \geq p_t^\lambda(2)$. Since $|p_t^\lambda(1) -$

$p_t^\lambda(2)| \le |p_t(1) - p_t(2)|$, we have that $\mathbf{p}_{2,t}^\lambda \mathbf{M}_{a,\epsilon}\mathbf{q}_t \le \mathbf{p}_{2,t}\mathbf{M}_{a,\epsilon}\mathbf{q}_t$. Thus we can conclude

$$\frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_{2,t}^\lambda \mathbf{M}_{a,\epsilon}\mathbf{q}_t \le \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_{2,t}\mathbf{M}_{a,\epsilon}\mathbf{q}_t \le \min_{\mathbf{p}\in\mathbb{P}_2} \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}\mathbf{M}_{a,\epsilon}\mathbf{q}_t + \Delta_{T,n}$$

where the second inequality follows from Theorem 2.10.

Condition (b) clearly holds, because $\mathbf{p}\mathbf{M}_{a,\epsilon}\mathbf{q} = M_{a,\epsilon}(3,\mathbf{q})$ for all $\mathbf{p} \in \mathbb{P}_{-2}$ and $\mathbf{q} \in \mathbb{Q}$.

Condition (c) clearly holds with $C = \frac{\lambda}{3}$.

To prove condition (d), it suffices to show that $\overline{p}_T^\lambda(3) \le \frac{\lambda}{3} + \frac{\Delta_{T,n}}{\delta_{\mathbf{M}}^v}$, where $\overline{p}_T^\lambda(3) \triangleq \frac{1}{T}\sum_{t=1}^{T} p_t^\lambda(3)$. Since $\overline{p}_T^\lambda(3) = (1-\lambda)\overline{p}_T(3) + \frac{\lambda}{3}$, we only need to show that $\overline{p}_T(3) \le \frac{\Delta_{T,n}}{\delta_{\mathbf{M}}^v}$.

Consider a repeated game in which the row player chooses strategies $\mathbf{p}'_1, \ldots, \mathbf{p}'_T$ according to the MW algorithm, and the column player chooses strategy $\mathbf{q}'_t = \arg\max_{\mathbf{q}\in\mathbb{Q}} \mathbf{p}'_t\mathbf{M}_{a,\epsilon}\mathbf{q}$ in each round t. We have

$$\mathbf{p}'_t\mathbf{M}_{a,\epsilon}\mathbf{q}'_t \ge \mathbf{p}'_t\mathbf{M}_{a,\epsilon}\mathbf{q}^{\text{lex}\,*} \ge v^*(1) + \delta_{\mathbf{M}}^v p'_t(3)$$

where the first inequality holds by the choice of $\mathbf{q}'_t$, and the second inequality holds because $v^*(3) = v^*(1) + \delta_{\mathbf{M}}^v$. Taking the average of both sides over all T rounds and applying Corollary 2.4 yields

$$\Delta_{T,n} \ge \delta_{\mathbf{M}}^v \overline{p}'_T(3)$$

If we can show $\overline{p}'_T(3) \ge \overline{p}_T(3)$, we are done. We will prove something stronger: $M_{a,\epsilon}(3,\mathbf{q}'_t) \le M_{a,\epsilon}(3,\mathbf{q}_t)$ for all rounds t (which implies that that $p'_t(3) \ge p_t(3)$ for all rounds t). For contradiction, let t be the first round where $M_{a,\epsilon}(3,\mathbf{q}'_t) > M_{a,\epsilon}(3,\mathbf{q}_t)$. Then it must be that

$$|p'_t(1) - p'_t(2)| < |p_t^\lambda(1) - p_t^\lambda(2)|$$

But by the choice of t, it must be that

$$|p_t(1) - p_t(2)| \leq |p'_t(1) - p'_t(2)|$$

And we have already argued that $|p^\lambda_t(1) - p^\lambda_t(2)| \leq |p_t(1) - p_t(2)|$, which is a contradiction. □

3.8 Other Related Work

Lexicographic optimal strategies are closely related to several other concepts in game-theoretic literature. The most similar is the notion of *proper equilibria* in n-player noncooperative games, introduced by Myerson [83]. Proper equilibria are themselves defined in terms of *ϵ-proper equilibria*: With respect to a particular assignment of strategies to the players, let us order each player's pure strategies from best to worst. If each player has been assigned a mixed strategy which places at most ϵ times as much weight on a worse pure strategy than on a better one, then this assignment of strategies is called an ϵ-proper equilibrium. A proper equilibrium is the limit of a sequence of ϵ-proper equilibria in which $\epsilon \to 0$. Myerson [83] showed that proper equilibria always exist, and they are a subset of the Nash equilibria. Later, van Damme [133] showed that if both players in a zero-sum game are using a lexicographic optimal strategy, then their strategies constitute a proper equilibrium — thus, in the case of two-player zero-sum games, these concepts are equivalent.

Other related game-theoretic concepts are *subgame perfect*, *sequential*, and *quasiperfect* equilibria. Like lexicographic optimality, these concepts are based on anticipating the possibility that an opponent will make a mistake. A good reference for these so-called *equilibrium refinements* is by van Damme [134].

Potters and Tijs [95] defined the *nucleolus* of a convex map, and showed that, for any two-plyaer zero-sum game, one can construct a convex map such that its

nucleolus coincides with the set of lexicographic optimal strategies.

As we have already discussed, Miltersen and Sorensen [79] were the first to describe a polynomial-time algorithm for computing lexicographic optimal strategies in two-player zero-sum games. Miltersen and Sorensen [80] improve their earlier algorithm so that it runs in time polynomial in the game's extensive form representation. For some games, the extensive form is much more compact than the normal form, but this is not always true. In particular, it is not the case for the games we described in Sections 2.1 and 2.3, or the game we will discuss at length in Chapter 5.

The importance of lexicographic optimal strategies, and related concepts, has long been recognized in applications. For example, Koller and Pfeffer [62] noted that their poker-playing application tends to give back "gifts" it receives from her opponent, i.e., it does not take advantage of mistakes.

McCracken and Bowling [76] and Johanson et al. [51] introduced the concepts of *safe* and *robust* strategies, respectively, which bear a resemblance to lexicographic optimal strategies. In their setting, one player builds a model of her opponent's behavior, and then calculates a best response to the model. Safe and robust strategies provide a way, derived from game-theoretic principles, to guard against the possibility that the opponent will deviate from the model.

3.9 Conclusion

In this chapter, we defined a useful refinement of the minimax/maximin optimal strategy concept for two-player zero-sum games. We also showed how no-regret algorithms, in particular the MW algorithm, are well-suited to computing these kinds of strategies. We will use these algorithms in the Chapter 5.

Chapter 4

Mimicking Approach to Apprenticeship Learning

In this chapter and Chapter 5, we present algorithms for a variant of reinforcement learning called *apprenticeship learning*, first introduced by Abbeel and Ng [1]. The basic idea underlying apprenticeship learning is that a learning agent, called the *apprentice*, is able to observe another agent, called the *mentor*, behaving in an environment. The mentor's behavior "teaches" the apprentice about the unknown rewards. The goal of the apprentice is to learn a *policy* — i.e., a concrete prescription of how to behave in the environment — that is at least as good as the mentor's policy, relative to the unknown rewards. This is a weaker requirement than the usual goal in reinforcement learning, which is to find a policy that maximizes reward.

Rewards are generally regarded as the *sine qua non* of reinforcement learning, and it is certainly difficult to imagine reinforcement learning without the "reinforcement" that rewards provide. However, merely stating the centrality of rewards to the framework does not avoid a difficult truth about them: in practice, specifying the rewards correctly is often very hard. Of course, in any application, some basic properties of the rewards will be obvious. For example, when driving a car, crashing should be a

low-reward event. When operating a robotic arm, grasping the target object should earn higher reward than missing it. And when playing a game of backgammon, the highest reward should be assigned to winning the game.

In the reinforcement learning framework, rewards are represented by real numbers, which means that one must select specific values to represent "high" and "low" rewards. Choosing the exact values for the rewards is a subtle and delicate design problem. When driving a car, should staying on the road be ten times more rewarding than avoiding a crash? Or one hundred times more rewarding? Unfortunately, the behavior learned by most reinforcement learning algorithms can be quite sensitive to the specific numerical values of the rewards. As a result, in practice, rewards are frequently tweaked and tuned to elicit the desired behavior. We present an illustration of this phenomenon in Section 4.1.

Sections 4.2 and 4.3 provide a detailed presentation of the apprenticeship learning framework. All existing algorithms for apprenticeship learning are based on mimicking the behavior of the mentor as closely as possible, and the algorithms we develop in this chapter do this as well. On the other hand, in Chapter 5 we explain how the game-solving techniques from Chapters 2 and 3 can be applied to apprenticeship learning.

In Section 4.4, we review the main algorithm due to Abbeel and Ng [1], and describe an improved version of the algorithm that is both simpler and has superior theoretical guarantees. Our algorithm is based on Blackwell's theory of approachability [10].

The algorithms of Section 4.4 have one major drawback; they assume the true rewards can be expressed as a linear combination of a set of known features. In many cases, even this knowledge of the rewards may be difficult to obtain. In Section 4.5, we remove this requirement, and describe an approach to apprenticeship learning that assumes almost no prior knowledge about the true rewards. The idea is to *reduce* the

apprenticeship learning problem to classification, one of the oldest and most well-studied problems in machine learning. The idea of reducing one learning problem to another is a powerful one, and was first proposed by Zadrozny et al. [147]. Using this approach, we show how any existing classification algorithm can be converted into an apprenticeship learning algorithm, with its theoretical guarantees carrying over into the new setting. The classification algorithm is used to mimic the behavior of the mentor.

4.1 Motivation

We have asserted that specifying rewards can be difficult. For an illustration of this phenomenon, consider the simple car driving simulator depicted in Figure 4.1. We will return to this simulator in Section 5.7. In this simulator, the agent is the driver of the dark-colored car, which is on a busy three-lane highway. Light-colored cars pass by the agent's car continually. The agent's car has three available speeds: "fast", "medium", and "slow".

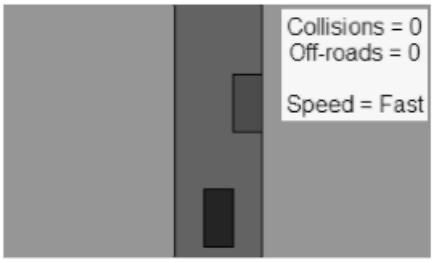

Figure 4.1:
Screenshot of car driving simulator.

How should the agent drive her car? Perhaps the most simple and natural way to describe the behavior we want is to use plain English: the agent should "drive as fast as possible while avoiding other cars". But how does one translate this goal into rewards? In practice, this is usually done in a fairly *ad hoc* fashion. As a first

attempt, let us try assigning a reward of +10 to each time step in which the agent's car is traveling at high speed, +5 for medium speed, and 0 for slow speed. Let us also assign a reward of −100 for colliding with other cars.

We applied a reinforcement learning algorithm called *value iteration* — reviewed in Section 4.2.1 — to learn a policy for the agent's car that maximizes reward. The first line of Table 4.1, denoted Attempt 1, describes the behavior of the learned policy. It is clear that something has gone wrong. The agent's car is driving fast and avoiding other cars, but spends all its time off-road. The problem, of course, is that we forgot to add a penalty for driving off-road.

	Speed	Collisions (per sec)	Off-roads (per sec)
Attempt 1	Fast	0	8.0
Attempt 2	Fast	0.5	0
Attempt 3	Medium	0	1.3
Attempt 4	Medium	0	0

Table 4.1:
Attempts to tune reward function.

So let us add such a penalty. We assigned a reward of −50 to driving off-road, and then re-learned a policy that maximizes reward. The results are given in Table 4.1, as Attempt 2. In this case, the agent's car stays on the road, but does not completely succeed in avoiding other cars. Evidently, the penalty for crashing is too small relative to the reward for driving fast.

The rest of Table 4.1 describes attempts to tweak the rewards to fix this problem. By Attempt 4, we have succeeding in learning a crash-free, on-road driving policy for the blue car. Because the environment of the car driving simulator is small and simple, we can manually verify that "medium" is indeed the fastest speed that the blue car can drive while ensuring that no crashes occur and the car stays on the road. In a more complicated environment, of course, it would be impossible to check this.

The illustration given above actually understates the difficulty of specifying ap-

propriate rewards. In the car driving simulator, it was fairly painless to tweak the rewards until the desired behavior was achieved. But suppose we had been doing this experimentation while driving a real car? Tweaking the rewards in such a situation could be disastrous.

Repeatedly adjusting rewards to correct for unintended behavior is a common experience when using reinforcement learning algorithms. This often happens when rewards are given for "intermediate" goals that can inadvertently cause the algorithm to ignore the actual goal. For example, in work by Randlov and Alstrom [103] on teaching an agent to ride a simulated bicycle, the agent only learned to ride the bicycle in a circle, because rewards were given for staying upright. Also, Andre and Teller [2] describe a soccer-playing robot which learned only to "vibrate" next to the ball, hitting it repeatedly, because rewards were given for contact with the ball.[1]

Apprenticeship learning is one approach to dealing with the difficulty of specifying the rewards in a reinforcement learning problem. Before describing this approach precisely, we will describe *Markov Decision Processes* in Section 4.2, which provide a mathematical foundation for reinforcement learning. Then, in Section 4.3, we will explain how the apprenticeship learning framework departs from a Markov Decision Process.

4.2 Markov Decision Processes

Reinforcement learning and Markov Decison Processes (MDPs) are so closely related that the terms are sometimes used interchangeably. Put simply, an MDP is the formal mathematical framework within which the problem of reinforcement learning is studied.[2] MDPs are used extensively in many fields besides machine learning, such as control theory and operations research. They are a natural formalism for study-

[1] These anecdotes were first related by Ng et al. [90].

[2] There are some alternative frameworks (e.g. Predictive State Representations [74]), but these have been developed relatively recently and have not been widely adopted.

ing problems in which an agent must make a sequence of decisions in an uncertain environment.

The main elements of an MDP are the *state space* $\mathcal{S}$, *action space* $\mathcal{A}$, *policies* Π, *transition function* θ, and *reward function* R. We will describe each of these in turn.

Each state $s \in \mathcal{S}$ describes a possible configuration of the environment. In our car driving simulator, a state of the environment is just a description of the positions and speeds of all the cars. Choosing an appropriate definition for the state space is itself a subtle design issue (Powell [96, p. 139] provides an extensive discussion). It can sometimes be difficult to know which details of the environment should be included in the state descriptor, and which should be abstracted away. Ideally, the state should contain all the information that an agent needs to predict the future evolution of the environment, so that, conditioned on the state, the future of the environment should be independent of its past. Put another way, the state should render the environment *Markovian* — hence the name "Markov Decision Process". The state space can be infinite, but unless we specify otherwise, the state space will be presumed to be finite, though possibly very large. Infinite state spaces will be considered in Section 4.5.

Each action $a \in \mathcal{A}$ is an option available to the agent for controlling its environment at a given moment. In our car driving simulator, the actions are: moving left and moving right, and speeding up and slowing down. Like the state space, we will assume that the set of actions is finite.

A policy $\pi \in \Pi$ is a prescription for an agent for controlling its environment. Policies link states to actions. A policy π can be *deterministic*, in which case $\pi : \mathcal{S} \to \mathcal{A}$ is a function mapping each state to a single action. A policy can also be *randomized*, in which case $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a function mapping each state-action pair to a probability. For randomized policies, we let $\pi(s, a)$ denote the probability, according to policy π, of taking action a in state s (observe that this notation is general enough to describe deterministic policies as well). We must have $\sum_{a \in \mathcal{A}} \pi(s, a) = 1$ for all states

$s \in \mathcal{S}$. For both the deterministic and randomized cases, we have assumed that π is *stationary*, which means that it does not vary with time. The MDP framework is flexible enough to allow for nonstationary policies, which can also be further divided into deterministic and randomized types. However, unless specified otherwise, we will consider only stationary policies (or mixtures of stationary policies; see Section 4.4). Nonstationary policies will be studied in Section 4.5.

The transition function $\theta : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ describes the dynamics of the environment, i.e., how the environment evolves from state to state, under a specific choice of action by the agent. The quantity $\theta(s, a, s')$ is the probability of transitioning to state s' when the agent takes action a in state s. We must have $\sum_{s' \in \mathcal{S}} \theta(s, a, s') = 1$ for all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$. Algorithms for reinforcement learning can be divided into two major categories: those which require that the transition function be given, and those which do not assume that it is given.[3] In this thesis, we will study both settings.

Finally, we come to the most crucial element of MDPs, which we have already discussed at length: the reward function. A reward function $R : \mathcal{S} \to \mathbb{R}$ maps each state to a real number. It is possible to generalize this definition to allow reward functions to depend jointly on both states and actions. Making this generalization is easy, but having a dependence on actions tends to clutter the analysis. Consequently, in much of reinforcement learning literature this generalization is ignored, and we will also follow this convention. We also let $R^{\max} = \max_{s \in \mathcal{S}} |R(s)|$ denote the largest reward magnitude.

Having described the main elements of an MDP, we can formalize the goal of reinforcement learning within the MDP framework. To do this, we first need to define the *value function* V^π for a policy π. The value function is defined in terms of

[3]Some authors take the position that the reinforcement learning algorithms *by definition* may not assume that the transition function is given, but this strict usage of terminology is not universally accepted.

all the elements of an MDP that we have described so far, and is given by

$$V^{\pi}(s) \triangleq E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \;\middle|\; s_0 = s, a_t \sim \pi(s_t, \cdot), s_{t+1} \sim \theta(s_t, a_t, \cdot)\right]$$

In other words, $V^{\pi}(s)$ is the expected cumulative discounted reward for following policy π when starting in state s. The *discount factor* $\gamma \in [0, 1)$ encodes the principle that rewards received sooner are worth more than rewards received later. The discount factor also offers the mathematical convenience of ensuring that the infinite sum in the definition of $V^{\pi}(s)$ is finite, as long as the reward function is bounded. The fact that $V^{\pi}(s)$ depends on rewards received infinitely far into the future is known as an *infinite horizon*. While $V^{\pi}(s)$ could have been defined using a finite horizon, tasks that take a long or indefinite amount of time to complete are often well-modeled using an infinite horizon plus a discount factor. We will use a finite horizon in Section 4.5.

Let $\alpha : \mathcal{S} \to [0, 1]$ denote an *initial state distribution*, where $\alpha(s) \geq 0$ for all states $s \in \mathcal{S}$ and $\sum_{s \in \mathcal{S}} \alpha(s) = 1$. The value of a policy π is defined to be $V(\pi) \triangleq \sum_{s \in \mathcal{S}} \alpha(s) V^{\pi}(s)$, and an *optimal policy* for the MDP is

$$\pi^* \triangleq \arg\max_{\pi \in \Pi} V(\pi) \tag{4.1}$$

We say that a policy π is ϵ-*optimal* if $V(\pi) \geq V(\pi^*) - \epsilon$.

We note that our definition of optimality is weaker than the definition most often used in literature on MDPs. Usually, an optimal policy π^* is defined to be one that satisfies

$$V^{\pi^*}(s) \geq V^{\pi}(s) \text{ for all } \pi \in \Pi \text{ and } s \in \mathcal{S} \tag{4.2}$$

Clearly, (4.2) is a stronger requirement than (4.1). Indeed, (4.2) is a very strong property for a policy to have. It is not obvious that a policy that maximizes the value function at every state simultaneously even exists. One of the most fundamental

results in the study of MDPs is that such a policy does exist, and moreover that it is both deterministic and stationary [7, 50].

For our purposes, the weaker definition of optimality given in (4.1) will suffice. One reason is that many applications have a well-known initial state distribution. Games like chess, backgammon, etc., for instance, are in this category. Another reason is that, when we introduce the apprenticeship learning framework in Section 4.3, this weaker definition will allow us to make weaker assumptions about the mentor. In particular, we will only need to assume that we can observe the mentor starting in a state drawn from the initial state distribution, and not in each state separately.

The usual goal in reinforcement learning is to *efficiently* find an approximately optimal policy in the MDP, using time that is polynomial in the quality of the approximation and the size of the MDP. We state this goal formally so we can easily compare it with the goal of apprenticeship learning that we present in the Section 4.3.

> **Goal of Reinforcement Learning:** Find an ϵ-optimal policy in $\mathrm{poly}(|\mathcal{S}|, |\mathcal{A}|, 1/\epsilon, R^{\max}/(1-\gamma))$ time.

4.2.1 Computing an Optimal Policy

The goal of reinforcement learning is to compute an optimal policy in an MDP, and a huge array of methods have been developed for this task. In this section we review several of the better-known reinforcement learning algorithms. These algorithms will be used as subroutines for the apprenticeship learning algorithms that we developed in later sections. All of these algorithms are designed to compute a policy that is optimal in the strong sense defined in (4.2) — i.e., a policy that maximizes the value function at every state simultaneously — and throughout this section we will use this definition of optimalty exclusively. Of course, any such policy is also optimal in the weaker sense defined in (4.1).

Value Iteration

The original algorithms for computing optimal policies require access to both the reward function R and the transition function θ of the MDP. The earliest such algorithm is *value iteration*, first proposed by Bellman [7], who also proved that every MDP has an optimal *deterministic* policy. Bellman's approach was to show that, if such a policy π^* exists, its value function V^{π^*} must satisfy

$$V^{\pi^*}(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \theta(s, a, s') V^{\pi^*}(s') \tag{4.3}$$

for all states $s \in \mathcal{S}$. Moreover, any real-valued function $V^* : \mathcal{S} \to \mathbb{R}$ satisfying (4.3) is the value function of an optimal deterministic policy π^*, and this policy can be recovered by setting

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \theta(s, a, s') V^*(s')$$

for each $s \in \mathcal{S}$. In other words, $\pi^*(s)$ is the action that realizes the maximum on the right-hand side of (4.3).

Let $\mathcal{V}$ be the set of all functions $V : \mathcal{S} \to \mathbb{R}$ such that $\max_{s \in \mathcal{S}} |V(s)| \leq \frac{R^{\max}}{1-\gamma}$. It is easy to show that the value function V^π of any policy π belongs to $\mathcal{V}$. Now define an operator $T^* : \mathcal{V} \to \mathcal{V}$ which, given a function $V \in \mathcal{V}$, returns a new function $T^*(V)$ that is obtained by plugging V into the right-hand side of (4.3) and returning the left-hand side (i.e., treating (4.3) as an assignment instead of an equation). Bellman proved that a solution to (4.3) always exists by showing that the operator T^* is a *contraction* on $\mathcal{V}$, and thus has a (unique) fixed point [110]; this fixed point is the solution V^* of (4.3). Appropriately, (4.3) is known as the *Bellman equation*.

Moreover, the operator T^* can used to estimate V^*; this is the value iteration algorithm. Starting at any function $V^1 \in \mathcal{V}$, value iteration produces a sequence of

functions $V^1, V^2, \ldots$ by setting $V^{i+1} = T^*(V^i)$. The contraction property of T^* can be used to show that the function V^n satisfies

$$|V^n(s) - V^*(s)| \leq 2R^{\max}\frac{\gamma^n}{1-\gamma}$$

for all states $s \in \mathcal{S}$.

Policy Iteration

While value iteration operates on value functions, the *policy iteration* algorithm operates directly on policies. The algorithm produces a sequence of policies $\pi_1, \pi_2, \ldots$ over the course of its operation. The policy π_1 is chosen arbitrarily, and for each policy π_i, the value function V^{π_i} is computed using any of the methods described in Section 4.2.2. If π_i is an optimal policy, then V^{π_i} satisfies the Bellman equation (4.3); in this case the algorithm terminates. If π_i is not an optimal policy, then the next policy π_{i+1} is obtained by setting

$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \theta(s, a, s') V^{\pi_i}(s')$$

for each state $s \in \mathcal{S}$. It is easy to show that $V^{\pi_{i+1}}(s) > V^{\pi_i}(s)$ for at least one state s, and therefore the algorithm cannot cycle. Moreover, since the number of deterministic policies is finite, the algorithm must eventually terminate. A more refined analysis due to Puterman [100] shows that policy iteration converges no more slowly than value iteration.

Despite the similar bounds on their worst-case running time, policy iteration is often much faster than value iteration in practice, and is widely-used in applications. In Section 5.7 we will discuss several experiments involving both value iteration and policy iteration.

Value iteration and policy iteration can be combined to form hybrid methods,

such as *modified policy iteration* [101], which interleaves the operation of the two algorithms. Also, we have thus far assumed that value iteration and policy iteration update *every* state in every iteration. In large environments, this can cause progress to be very slow. *Asynchronous policy iteration* algorithms [8] update only a few states every iteration; *prioritized sweeping* [82, 93] is one method for intelligently selecting the states to update.

Linear Programming

It was observed quite early in the study of MDPs that the problem of computing an optimal policy can be formulated as a linear program [20]. The linear program has the following form:

$$\min_V \sum_{s \in \mathcal{S}} V(s)$$
$$V(s) \geq R(s) + \gamma \sum_{s \in \mathcal{S}} \theta(s, a, s') V(s) \quad \text{for all } s \in \mathcal{S},\ a \in \mathcal{A}$$

Note that the optimal value function V^*, which is a solution to (4.3), is also a feasible solution for this linear program. Moreover, if a feasible solution V to the linear program differs from V^*, then there must be some state $s \in \mathcal{S}$ such that all the constraints in which $V(s)$ appears on the left-hand side are loose, which implies that V is not an optimal solution. These facts together prove the correctness of the linear program.

We will use the dual of this linear program in Section 5.6, where we describe an efficient method for solving the apprenticeship learning problem.

Model-Free Algorithms

All the reinforcement learning algorithms that we have reviewed thus far require an explicit description of the environment, in the form of the reward function R and tran-

sition function θ. It is often unrealistic to assume that such a description is available, and much of reinforcement learning research has been driven by an interest in removing this requirement. Indeed, as we mentioned in Section 4.2, some authors argue that only algorithms that do not require an explicit description of the environment can be properly called "reinforcement learning" algorithms.

The algorithms we describe below do not operate on a description of the environment, but *interact* with the environment directly. Their experience with the environment takes the form of one continuous trajectory $\{(s_t, r_t, a_t)\}_{t=1}^{\infty}$. When one of these algorithms visits state s_t, it observes a reward $r_t = R(s_t)$, takes some action a_t, and is then sent to a new state s_{t+1} according to the distribution $\theta(s_t, a_t, \cdot)$. As usual, the aim of the algorithm is to learn an optimal policy π^*.

Reinforcement learning algorithms for this setting can be divided into two major types. The first type are *model-based* algorithms, which construct an approximation, or model, of the true MDP by estimating the unknown functions R and θ, and then use a method like value iteration to learn an optimal policy in the model. We will describe model-based algorithms in the next subsection, where we review algorithms that have polynomial-time bounds on their running time. In this subsection, we describe *model-free* algorithms, which make no attempt to explicitly model the dynamics of the environment, but instead learn an optimal policy via a more direct approach.

Q-learning [142] is one of the most popular and elegant model-free reinforcement learning algorithms. It belongs to a family of model-free algorithms that estimate the *state-action value function* Q^π of a policy π, defined

$$Q^\pi(s, a) \triangleq R(s) + \gamma \sum_{s' \in \mathcal{S}} \theta(s, a, s') V^\pi(s')$$

The objective of Q-learning is to estimate Q^{π^*}, the state-action value function of the optimal policy π^*. Since $V^{\pi^*}(s) = \max_{a \in \mathcal{A}} Q^{\pi^*}(s, a)$, the policy π^* can be recovered

from Q^{π^*} by setting $\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi^*}(s, a)$. The Q-learning algorithm estimates Q^{π^*} by maintaining a function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ whose values are updated at each time step t as follows

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

where $\alpha_t \in \mathbb{R}$ is called the *learning rate* at time t. If each state-action pair (s, a) is visited infinitely often by the sequence $\{(s_t, r_t, a_t)\}_{t=1}^{\infty}$, and if

$$\sum_{t=1}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=1}^{\infty} (\alpha_t)^2 < \infty$$

then Q converges to Q^{π^*} with probability 1 [143]. Note that the policy which generates the experience trajectory $\{(s_t, r_t, a_t)\}_{t=1}^{\infty}$, called the *exploration policy*, need not be an optimal policy; this makes Q-learning an *off-policy* method. Other model-free algorithms, such as SARSA [111], are *on-policy* methods. They work by estimating the state-action value function Q^{π} of the exploration policy π, and then slowly modify that policy until it is optimal.

Polynomial-Time Methods

Although the methods described in the previous subsubsection do not require an explicit description of the environment, they are only guaranteed to converge to an optimal policy after a finite, but possibly very large, number of time steps. The E^3 algorithm [56] (which stands for "Explicit Explore or Exploit") was the first reinforcement learning algorithm with a *polynomial*-bound on its convergence time. The more recent R-MAX algorithm [13] can be viewed as a simpler and more intuitive version of E^3, so we review its operation here.

As the R-MAX algorithm interacts with the environment, it maintains a model of the environment based on its experience so far, and always follows an optimal policy

in that model. Among the states the algorithm has visited, it keeps track of which are "known" (i.e., have been visited a sufficient number of times), and which are not. If a state is not known, its reward in the model is set to $R^{\max}$, the maximum reward of any state (hence the name of the algorithm). This encourages the algorithm to visit states it has not visited very often, and yields the following measure of progress: At every time step, either the R-MAX algorithm is following a near-optimal policy, or it is efficiently learning a better model of the environment.

Every reinforcement learning algorithm which is not given a description of its environment is faced with the "exploration vs. exploitation" dilemma: At every time step, the algorithm must decide whether to take an action which leads to large reward in its current understanding of the environment, or to take an action which improves its understanding of the environment. Many reinforcement learning algorithms resolve this dilemma in fundamentally the same way that the R-MAX algorithm does, by maintaining "optimism in the face of uncertainty".

Both E^3 and R-MAX are model-based algorithms, as are most reinforcement learning algorithms with polynomial bounds on their convergence time. But a model-based approach is not strictly necessary; the Delayed Q-learning algorithm [123] is a variant of Q-learning which provably converges to an optimal policy in a polynomial number of time steps.

Approximate Methods

All of the reinforcement learning algorithms we have discussed so far assume that the state space is finite. In practice, this can be a very limiting assumption. For example, many real-world domains are most naturally modeled by a state space that is a subset of $\mathbb{R}^n$. However, when the state space is infinite, no algorithm can be guaranteed to compute an optimal policy unless further assumptions are made. Nonetheless, the most active research area in traditional reinforcement learning is in developing

algorithms that work well in very large or infinite state spaces. We note that nearly all of the apprenticeship learning algorithms that we present in later sections are really meta-algorithms that can use any of these methods as a subroutine, and therefore inherit any desirable properties or guarantees they may possess.

A comprehensive review of these methods is well outside the scope of this thesis. Both Sutton and Barto [125] and Powell [96] offer excellent surveys; here we will just touch upon some of the basic ideas. The most common approach is *function approximation*, where the goal is to learn an approximation $V_{\boldsymbol{\theta}}$ or $Q_{\boldsymbol{\theta}}$ of the true value functions V or Q, parameterized by some low-dimensional vector $\boldsymbol{\theta}$. Instead of updating $V_{\boldsymbol{\theta}}$ or $Q_{\boldsymbol{\theta}}$ at individual points in their domain, function approximation methods update the components of $\boldsymbol{\theta}$ until they converge to a parameter $\boldsymbol{\theta}^*$ corresponding to an (approximately) optimal value function. Another approach is to assume that a near-optimal policy can be expressed in parameterized form $\pi_{\boldsymbol{\theta}}$. *Policy iteration* methods follow the gradient of the value function with respect to $\boldsymbol{\theta}$ until they reach an (approximately) optimal policy $\pi_{\boldsymbol{\theta}^*}$.

4.2.2 Computing Value of a Policy

In the course of computing an optimal policy, many reinforcement learning algorithms require the ability to compute the value function of a particular policy (policy iteration is one such algorithm). This step is called *policy evaluation*. For any policy π, its value function V^π must satisfy

$$V^\pi(s) = R(s) + \gamma \sum_{a \in \mathcal{A}, s' \in \mathcal{S}} \pi(s,a)\theta(s,a,s')V^\pi(s') \tag{4.4}$$

for all $s \in \mathcal{S}$. This is easy to prove by simply "unrolling" the definition of V^π, as follows:

$$\begin{aligned}
V^\pi(s) &= E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \;\middle|\; s_0 = s, a_t \sim \pi(s_t, \cdot), s_{t+1} \sim \theta(s_t, a_t, \cdot)\right] \\
&= R(s) + E\left[\sum_{t=0}^{\infty} \gamma^{t+1} R(s_{t+1}) \;\middle|\; s_0 = s, a_t \sim \pi(s_t, \cdot), s_{t+1} \sim \theta(s_t, a_t, \cdot)\right] \\
&= R(s) + \gamma E\left[\sum_{t=0}^{\infty} \gamma^t R(s_{t+1}) \;\middle|\; s_0 = s, a_t \sim \pi(s_t, \cdot), s_{t+1} \sim \theta(s_t, a_t, \cdot)\right] \\
&= R(s) + \gamma \sum_{a \in \mathcal{A}, s' \in \mathcal{S}} \pi(s, a)\theta(s, a, s') E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \;\middle|\; s_0 = s', a_t \sim \pi(s_t, \cdot), s_{t+1} \sim \theta(s_t, a_t, \cdot)\right] \\
&= R(s) + \gamma \sum_{a \in \mathcal{A}, s' \in \mathcal{S}} \pi(s, a)\theta(s, a, s') V^\pi(s')
\end{aligned}$$

Note that (4.4) closely resembles the Bellman equation in (4.3), and in fact (4.4) is also sometimes called the Bellman equation. If we regard each $V^\pi(s)$ as an unknown variable, then (4.4) defines a system of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns, which can be solved using a wide variety of techniques.

Alternatively, we can compute V^π by using repeated application of a contraction mapping, just like we did in the value iteration algorithm. Define an operator $T^\pi : \mathcal{V} \to \mathcal{V}$ which, given a function $V \in \mathcal{V}$, returns a new function $T^\pi(V)$ that is obtained by plugging V into the right-hand side of (4.4) and returning the left-hand side. T^π is a contraction, and repeated application of T^π converges to its unique fixed point V^π.

When a description of the environment is unavailable, the model-free TD(λ) algorithm [124] can be used to compute the value function of a policy. The SARSA algorithm is based on TD(λ).

The methods described above compute the value function V^π of a policy π, but most of the apprenticeship learning algorithms we describe in later sections only require the value of a policy $V(\pi)$ (recall that $V(\pi)$ is the expected value of $V^\pi(s)$

when the state s is drawn from the initial state distribution). Computing an estimate of $V(\pi)$ is as easy as estimating the value of random variable from independent samples: Simply execute the policy π in the environment for several trials, starting from the initial state distribution in each trial, and calculate the average cumulative discounted reward collected over all trials. Of course, the quality of this estimate will depend on both the number and duration of the trials. We will have more to say about this method of estimating $V(\pi)$ in Section 5.3. In Section 5.6, we describe another way to compute the value of a policy $V(\pi)$, which is based on the dual of the linear program that can be used to compute an optimal policy.

4.3 Apprenticeship Learning Framework

As we illustrated in Section 4.1, despite the centrality of rewards to the reinforcement learning framework, specifying the true reward function can be a difficult task. Abbeel and Ng [1]'s development of apprenticeship learning was motivated by the observation that, although the reward function may be difficult to specify, demonstrations of good behavior by a mentor are often plentiful. Therefore, by observing such a mentor, one can infer information about the true reward function without needing to specify it.

The apprenticeship framework is nearly identical to an MDP, except that the true reward function R is *unknown*. A *mentor policy* π_E can be observed executing in the environment, and the observations are in the form of m independent *trajectories*. A trajectory is just a sequence of states visited by the mentor during an interaction with the environment. We are now in a position to state the goal of apprenticeship learning.

Goal of Apprenticeship Learning: Find a policy π_A, called the *ap-*

prentice policy, such that

$$V(\pi_A) \geq V(\pi_E) - \epsilon$$

in $\text{poly}(|\mathcal{S}|, |\mathcal{A}|, k, 1/\epsilon, R^{\max}/(1-\gamma))$ time and using $\text{poly}(k, 1/\epsilon, R^{\max}/(1-\gamma))$ trajectories from the mentor, where the value of a policy is calculated with respect to the *unknown* reward function.

So apprenticeship learning is both easier and more difficult than traditional reinforcement learning. It is easier in the sense that the learned policy must only be nearly as good as the mentor policy, but not necessarily as good as an optimal policy. It is more difficult because the true reward function is unknown.

In the rest of this chapter and Chapter 5, we describe various algorithms for apprenticeship learning, each of which rely on different assumptions about the problem. In Section 4.4 and Chapter 5, we assume that the true reward function belongs to a restricted class — namely, it can be expressed as a linear combination of a set of known features. In Section 4.5, we assume we have access to a good classification algorithm, which we use to solve the apprenticeship learning problem.

4.4 Feature-Matching Algorithms

In this section, we present apprenticeship learning algorithms that are based on the idea of "feature-matching". The original apprenticeship learning algorithms due to Abbeel and Ng [1] are also based on this idea.

The idea is the following: Even when a reward function is difficult to describe exactly, it is usually easy to specify what the reward function must depend on. For example, when a person drives a car, the rewards that she is maximizing depend on just a few key factors: the speed of the car, the position of other cars, the underlying terrain, etc. What is unclear, however, is how the rewards encode the trade-offs

among these various factors. For example, exactly how much more should the driver prefer traveling fast over avoiding other cars?

With these observations in mind, in this section the unknown reward function R is assumed to have the form

$$R(s) = \mathbf{w}^* \cdot \boldsymbol{\phi}(s)$$

Here the *feature function* $\boldsymbol{\phi} : \mathcal{S} \to [0,1]^k$ is known, but the *weight vector* $\mathbf{w}^* \in \mathbb{R}^k$, where $\|\mathbf{w}^*\|_1 \leq 1$, is unknown. The bounds on the magnitude of the features and weight vector are needed in order to derive meaningful performance guarantees for apprenticeship learning algorithms. Clearly $R^{\max} \leq 1$ under these assumptions.

In the rest of this section, we will describe apprenticeship learning algorithms that are based on matching the behavior of the mentor with respect to the features. Before presenting these algorithms, we describe some additional components of the apprenticeship learning framework.

Feature Expectations

An object which plays a key role in our analysis is the *feature expectations* function:

$$\boldsymbol{\mu}(\pi) \triangleq E\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi}(s_t) \;\middle|\; s_0 \sim \alpha(\cdot), a_t \sim \pi(s_t, \cdot), s_{t+1} \sim \theta(s_t, a_t, \cdot)\right]$$

From its definition, it should be clear that "feature expectations" is a (somewhat misleading) abbreviation for "expected, cumulative, discounted feature values." Importantly, since $R(s) = \mathbf{w}^* \cdot \boldsymbol{\phi}(s)$, we have

$$V(\pi) = \mathbf{w}^* \cdot \boldsymbol{\mu}(\pi) \tag{4.5}$$

by linearity of expectation. We will repeatedly make use of the following fact about feature expectations: For any policy π, we have

$$\boldsymbol{\mu}(\pi) \in \left[\frac{1}{1-\gamma} \min_i \min_{s \in \mathcal{S}} \phi_i(s), \frac{1}{1-\gamma} \max_i \max_{s \in \mathcal{S}} \phi_i(s)\right]^k \tag{4.6}$$

by the convergence of the geometric series in the definition of $\boldsymbol{\mu}(\pi)$. Finally, let $\boldsymbol{\mu}_E = \boldsymbol{\mu}(\pi_E)$ denote the mentor's feature expectations.

Mixed Policies

To ease the apprenticeship learning task, in this section we will not require that the apprentice policy be a simple stationary policy, but instead allow it to be a *mixed policy*. A mixed policy $\boldsymbol{\psi}$ is described by a distribution over Π_D, the set of all *deterministic* stationary policies. Because Π_D is finite (though extremely large, since $|\Pi_D| = |\mathcal{A}|^{|\mathcal{S}|}$), we can fix a numbering of the policies in Π_D, which we denote $\pi^1, \ldots, \pi^{|\Pi_D|}$. This allows us to treat $\boldsymbol{\psi}$ as a vector, where $\psi(i)$ is the probability assigned to π^i. A mixed policy $\boldsymbol{\psi}$ is executed by randomly selecting the policy $\pi^i \in \Pi_D$ at time 0 with probability $\psi(i)$, and exclusively following π^i thereafter.

The value of a mixed policy is equal to the expected value of the stationary policies that constitute it:

$$V(\boldsymbol{\psi}) = E_{i \sim \boldsymbol{\psi}}\left[V(\pi^i)\right] = \sum_{i=1}^{|\Pi_D|} \psi(i) V(\pi^i) \tag{4.7}$$

The feature expectations of a mixed policy can be computed in the same way:

$$\boldsymbol{\mu}(\boldsymbol{\psi}) = E_{i \sim \boldsymbol{\psi}}\left[\boldsymbol{\mu}(\pi^i)\right] = \sum_{i=1}^{|\Pi_D|} \psi(i) \boldsymbol{\mu}(\pi^i) \tag{4.8}$$

Both (4.7) and (4.8) are easy consequences of the linearity of expectation, as is the following analogue of (4.5):

$$V(\boldsymbol{\psi}) = \mathbf{w}^* \cdot \boldsymbol{\mu}(\boldsymbol{\psi}) \tag{4.9}$$

Also note that mixed policies do not have any advantage over stationary policies in terms of value: if π^* is an optimal stationary policy, and $\boldsymbol{\psi}^*$ is an optimal mixed policy, then $V(\boldsymbol{\psi}^*) = V(\pi^*)$. Again, this is due to the linearity of expectation.

4.4.1 Projection Algorithm

We now review one of the original algorithms for apprenticeship learning, due to Abbeel and Ng [1]. We will provide a fairly detailed description and analysis, to ease the comparison with the new algorithms we present in later sections and chapters. Abbeel and Ng [1] actually described two apprenticeship learning algorithms. Both have the same theoretical guarantees, but the so-called *Projection* algorithm is simpler, and is also slightly faster in experimental studies, so we focus on it here.

The Projection algorithm is given in Algorithm 4.1, and a schematic depiction of the operation of the algorithm is given in Figure 4.2. In each iteration t, the Projection algorithm computes an optimal policy π_t for a reward function $R_t = \mathbf{w}_t \cdot \boldsymbol{\phi}$, where $\mathbf{w}_t$ is a weight vector updated each round by the algorithm. As Figure 4.2 illustrates, when the Projection algorithm ends, the point $\boldsymbol{\mu}_E$ is close to the convex hull of $\boldsymbol{\mu}(\pi_1), \ldots, \boldsymbol{\mu}(\pi_T)$. Moreover, $\boldsymbol{\mu}(\tilde{\boldsymbol{\psi}}_T)$ is the closest point in that convex hull to $\boldsymbol{\mu}_E$, where $\tilde{\boldsymbol{\psi}}_T$ is the mixed policy output by the algorithm. Consequently, Abbeel and Ng [1] are able to obtain the following result.

Theorem 4.1 ([1])**.** *Let $\tilde{\boldsymbol{\psi}}_T$ be the mixed policy output by the Projection algorithm. Then*

$$\left\| \boldsymbol{\mu}(\tilde{\boldsymbol{\psi}}_T) - \boldsymbol{\mu}_E \right\|_2 \leq O\left(\sqrt{\frac{k \log k}{T(1-\gamma)^2}} \right)$$

Theorem 4.1 can be used to prove that the mixed policy output by the Projection algorithm satisfies the goal of apprenticeship learning.

Corollary 4.2 ([1])**.** *Let $\tilde{\boldsymbol{\psi}}_T$ be the mixed policy output by the Projection algorithm.*

Algorithm 4.1 Projection Algorithm [1]

Given: MDP without a reward function, feature function $\boldsymbol{\phi}$, mentor's feature expectations $\boldsymbol{\mu}_E$, parameter T.
Choose any $\mathbf{w}_1$ such that $\|\mathbf{w}_1\|_1 \leq 1$.
for $t = 1, \ldots, T$ **do**
  Let $\pi_t = \arg\max_{\pi \in \Pi_D} \mathbf{w}_t \cdot \boldsymbol{\mu}(\pi)$.
  if t = 1 **then**
    Let $\boldsymbol{\mu}'_t = \boldsymbol{\mu}(\pi_t)$
  else
    Let $\boldsymbol{\mu}'_t = \boldsymbol{\mu}'_{t-1} + \frac{(\boldsymbol{\mu}(\pi_t) - \boldsymbol{\mu}'_{t-1}) \cdot (\boldsymbol{\mu}_E - \boldsymbol{\mu}'_{t-1})}{(\boldsymbol{\mu}(\pi_t) - \boldsymbol{\mu}'_{t-1}) \cdot (\boldsymbol{\mu}(\pi_t) - \boldsymbol{\mu}'_{t-1})} (\boldsymbol{\mu}(\pi_t) - \boldsymbol{\mu}'_{t-1})$
    (This is the projection of $\boldsymbol{\mu}_E$ onto the line through $\boldsymbol{\mu}'_{t-1}$ and $\boldsymbol{\mu}(\pi_t)$.)
  end if
  Let $\mathbf{w}_{t+1} = \boldsymbol{\mu}_E - \boldsymbol{\mu}'_t$.
end for
Find a solution $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_T$ to this quadratic program:

$$\min_{\lambda_1, \ldots, \lambda_T} \|\boldsymbol{\mu}_E - \boldsymbol{\mu}\|_2 \text{ such that } \boldsymbol{\mu} = \sum_{t=1}^{T} \lambda_t \boldsymbol{\mu}(\pi_t), \;\; \sum_{t=1}^{T} \lambda_t = 1, \;\; \lambda_t \geq 0$$

(The point $\sum_{t=1}^{T} \tilde{\lambda}_t \boldsymbol{\mu}(\pi_t)$ is the projection of $\boldsymbol{\mu}_E$ onto the convex hull of $\boldsymbol{\mu}(\pi_1), \ldots, \boldsymbol{\mu}(\pi_T)$.)
Return: The mixed policy $\tilde{\boldsymbol{\psi}}_T$ which assigns probability $\tilde{\lambda}_t$ to each policy π_t.

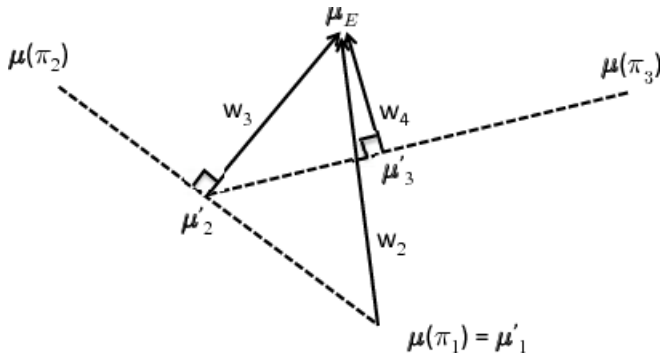

Figure 4.2: First several iterations of the Projection algorithm.

Then

$$\left|V(\tilde{\boldsymbol{\psi}}_T) - V(\pi_E)\right| \leq O\left(\sqrt{\frac{k \log k}{T(1-\gamma)^2}}\right) \tag{4.10}$$

Proof. We have

$$\begin{aligned}
\left|V(\tilde{\boldsymbol{\psi}}_T) - V(\pi_E)\right| &= \left|\mathbf{w}^* \cdot \boldsymbol{\mu}(\tilde{\boldsymbol{\psi}}_T) - \mathbf{w}^* \cdot \boldsymbol{\mu}_E\right| && ((4.9) \text{ and } (4.5)) \\
&\leq \|\mathbf{w}^*\|_2 \left\|\boldsymbol{\mu}(\tilde{\boldsymbol{\psi}}_T) - \boldsymbol{\mu}_E\right\|_2 && \text{(Cauchy-Schwarz)} \\
&\leq \left\|\boldsymbol{\mu}(\tilde{\boldsymbol{\psi}}_T) - \boldsymbol{\mu}_E\right\|_2 && (\|\mathbf{w}^*\|_2 \leq \|\mathbf{w}^*\|_1 \leq 1) \\
&\leq O\left(\sqrt{\frac{k \log k}{T(1-\gamma)^2}}\right) && \text{(Theorem 4.1)}
\end{aligned}$$

□

The guarantee in (4.10) implies that the values of the mentor's policy and the mixed policy output by the Projection algorithm differ by at most $O\left(\sqrt{\frac{k \log k}{T(1-\gamma)^2}}\right)$.

4.4.2 Blackwell Algorithm

One of the main drawbacks of the Projection algorithm is that it requires a quadratic program (QP) solver [12], which is used in a post-processing step to find the projection of a point onto a convex set. Generally, solving a QP is an expensive and complicated procedure. In this section, we will describe a modification to the Projec-

tion algorithm that makes this post-processing step unnecessary, and also makes the algorithm simpler and faster as part of the bargain. We call this new algorithm the *Blackwell* algorithm, because it is based on the well-known *Blackwell approachability theorem* [10].

Blackwell's theorem is a statement about *vector-payoff* repeated games, which are generalizations of zero-sum repeated games. In a zero-sum repeated game, as we explained in Section 2.1, the scalar payoff in each round is determined by the players' choice of strategies in that round, and the row player's goal is to minimize the average payoff over all rounds, while the column player's goal is to maximize it. In a vector-payoff repeated game, the payoff of the game is a vector instead of a scalar. The game is defined by a set S, and the row player's goal is to have the average payoff vector over all rounds be inside S, while the column player's goal is for it to be outside S. A set S is *approachable* in a vector-payoff repeated game if there exists a strategy-choosing algorithm for the row player which forces (asymptotically) the average payoff vector to be inside S, no matter what the column player does. Blackwell's approachability theorem states that a closed, convex set S is approachable if and only if every half-space containing S is approachable.

The proof of Blackwell's theorem is constructive; it actually describes the algorithm that ensures that the average payoff vector is asymptotically inside S. Algorithm 4.2 is essentially this algorithm applied to a vector-payoff repeated game in which the set $S = \{\boldsymbol{\mu}_E\}$. A schematic depiction of the operation of the algorithm is given in Figure 4.3. Just as in the Projection algorithm, in each iteration t, the Blackwell algorithm computes an optimal policy π_t for the reward function $R_t = \mathbf{w}_t \cdot \boldsymbol{\phi}$, although the weight vector $\mathbf{w}_t$ is updated differently. Figure 4.3 illustrates that, when the Blackwell algorithm ends, the point $\boldsymbol{\mu}_E$ is close to the *uniform average* of $\boldsymbol{\mu}(\pi_1), \ldots, \boldsymbol{\mu}(\pi_T)$. The next theorem proves this fact. Its proof is based on Blackwell's approachability theorem, although our proof more closely follows the presentation by

Cesa-Bianchi and Lugosi [16].

Algorithm 4.2 Blackwell Algorithm

Given: MDP without a reward function, feature function $\boldsymbol{\phi}$, mentor's feature expectations $\boldsymbol{\mu}_E$, parameter T.
Choose any $\mathbf{w}_1$ such that $\|\mathbf{w}_1\|_1 \leq 1$.
for $t = 1, \ldots, T$ **do**
    Let $\pi_t = \arg\max_{\pi \in \Pi_D} \mathbf{w}_t \cdot \boldsymbol{\mu}(\pi)$.
    Let $\overline{\boldsymbol{\mu}}_t = \frac{1}{t} \sum_{t'=1}^{t} \boldsymbol{\mu}(\pi_{t'})$.
    Let $\mathbf{w}_{t+1} = \boldsymbol{\mu}_E - \overline{\boldsymbol{\mu}}_t$.
end for
Return: The mixed policy $\overline{\boldsymbol{\psi}}_T$ given by the uniform distribution on $\pi_1, \ldots, \pi_T$.

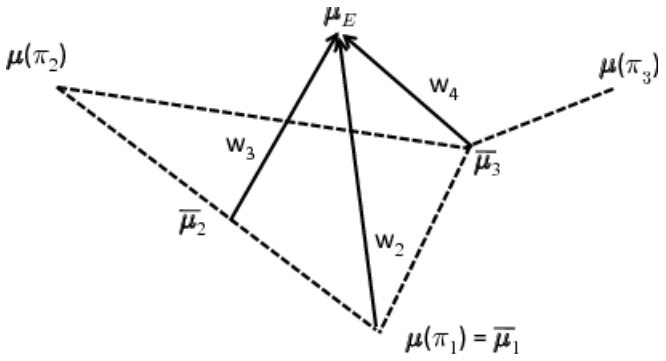

Figure 4.3: First several iterations of the Blackwell algorithm.

Theorem 4.3. *Let $\overline{\boldsymbol{\psi}}_T$ be the mixed policy output by the Blackwell algorithm. Then*

$$\left\|\boldsymbol{\mu}(\overline{\boldsymbol{\psi}}_T) - \boldsymbol{\mu}_E\right\|_2 \leq \sqrt{\frac{k}{T(1-\gamma)^2}} \tag{4.11}$$

Proof. For convenience, let $\boldsymbol{\mu}_t = \boldsymbol{\mu}(\pi_t)$, so that $\overline{\boldsymbol{\mu}}_t = \frac{1}{t} \sum_{t'=1}^{t} \boldsymbol{\mu}_{t'}$, and also let $\overline{\boldsymbol{\mu}}_0 = \overline{\boldsymbol{\mu}}_1$.

Since $\boldsymbol{\phi}(s) \in [0,1]^k$, we have that $\boldsymbol{\mu}(\pi) \in [0, \frac{1}{1-\gamma}]^k$ for any policy π, by (4.6). Therefore $\|\boldsymbol{\mu}_t - \boldsymbol{\mu}_E\|_\infty \leq \frac{1}{1-\gamma}$, which means that

$$\|\boldsymbol{\mu}_t - \boldsymbol{\mu}_E\|_2 \leq \sqrt{\frac{k}{(1-\gamma)^2}} \tag{4.12}$$

for all $t \in \{1, \ldots, T\}$. Now consider

$$\begin{aligned}
\|\overline{\boldsymbol{\mu}}_t - \boldsymbol{\mu}_E\|_2^2 &= \left\| \frac{t-1}{t} \overline{\boldsymbol{\mu}}_{t-1} + \frac{1}{t} \boldsymbol{\mu}_t - \boldsymbol{\mu}_E \right\|_2^2 \\
&= \left\| \frac{t-1}{t} \left(\overline{\boldsymbol{\mu}}_{t-1} - \boldsymbol{\mu}_E \right) + \frac{1}{t} \left(\boldsymbol{\mu}_t - \boldsymbol{\mu}_E \right) \right\|_2^2 \\
&= \left(\frac{t-1}{t} \right)^2 \left\| \overline{\boldsymbol{\mu}}_{t-1} - \boldsymbol{\mu}_E \right\|_2^2 + 2 \frac{t-1}{t^2} \left(\overline{\boldsymbol{\mu}}_{t-1} - \boldsymbol{\mu}_E \right) \cdot \left(\boldsymbol{\mu}_t - \boldsymbol{\mu}_E \right) + \frac{1}{t^2} \left\| \boldsymbol{\mu}_t - \boldsymbol{\mu}_E \right\|_2^2
\end{aligned}$$

Applying the inequality in (4.12), multiplying both sides by t^2, and rearranging yields

$$t^2 \left\| \overline{\boldsymbol{\mu}}_t - \boldsymbol{\mu}_E \right\|_2^2 - (t-1)^2 \left\| \overline{\boldsymbol{\mu}}_{t-1} - \boldsymbol{\mu}_E \right\|_2^2 \leq 2(t-1) \left(\overline{\boldsymbol{\mu}}_{t-1} - \boldsymbol{\mu}_E \right) \cdot \left(\boldsymbol{\mu}_t - \boldsymbol{\mu}_E \right) + \frac{k}{(1-\gamma)^2}$$

Now sum both sides over $t = 1, \ldots, T$. The left-hand side telescopes, yielding

$$T^2 \left\| \overline{\boldsymbol{\mu}}_T - \boldsymbol{\mu}_E \right\|_2^2 \leq 2 \sum_{t=1}^{T} (t-1) \left(\overline{\boldsymbol{\mu}}_{t-1} - \boldsymbol{\mu}_E \right) \cdot \left(\boldsymbol{\mu}_t - \boldsymbol{\mu}_E \right) + \frac{Tk}{(1-\gamma)^2}$$

Dividing both sides by T^2 yields

$$\left\| \overline{\boldsymbol{\mu}}_T - \boldsymbol{\mu}_E \right\|_2^2 \leq -\frac{2}{T^2} \sum_{t=1}^{T} (t-1) \mathbf{w}_t \cdot \left(\boldsymbol{\mu}_t - \boldsymbol{\mu}_E \right) + \frac{k}{T(1-\gamma)^2}$$

where we used the fact that $\mathbf{w}_t = \boldsymbol{\mu}_E - \overline{\boldsymbol{\mu}}_{t-1}$ for $t > 1$. The choice of π_t implies that $\mathbf{w}_t \cdot \boldsymbol{\mu}_t \geq \mathbf{w}_t \cdot \boldsymbol{\mu}_E$, which means each term in the sum on the right-hand side is nonnegative, and the entire sum can be dropped. Taking square roots and noting that $\overline{\boldsymbol{\mu}}_T = \boldsymbol{\mu}(\overline{\boldsymbol{\psi}}_T)$ by (4.8) implies the theorem. □

We can use Theorem 4.3 to prove that the value of the mixed policy output by the Blackwell algorithm is close to the value of the mentor's policy.

Corollary 4.4. *Let $\overline{\boldsymbol{\psi}}_T$ be the mixed policy output by the Blackwell algorithm. Then*

$$\left| V(\overline{\boldsymbol{\psi}}_T) - V(\pi_E) \right| \leq \sqrt{\frac{k}{T(1-\gamma)^2}} \tag{4.13}$$

Proof. The proof is identical to the proof of Corollary 4.2, except in the last line we use Theorem 4.3 instead of Theorem 4.1. □

Comparing Corollary 4.4 to Corollary 4.2, we see that the Blackwell algorithm, in addition to being substantially simpler, enjoys a rate of convergence that is $\log k$ times faster than the Projection algorithm.

Completing the Algorithm Description and Analysis

The alert reader will note that both the Projection and Blackwell algorithms, as currently presented, are underspecified. By examining the descriptions given in Algorithms 4.1 and 4.2, it is evident that both algorithms require access to subroutines that can compute the following:

- The optimal policy for a given reward function.
- The feature expectations of a given policy.
- The feature expectations $\boldsymbol{\mu}_E$ of the mentor's policy π_E.

Importantly, these quantities can all be efficiently estimated. In Section 4.2.1 we reviewed several classical algorithms for computing the optimal policy in an MDP. Also, note that computing the feature expectations of a policy is equivalent to computing the value of a policy with respect to several different reward functions (regarding each feature as a separate "reward function"), and in Section 4.2.2 we described algorithms for computing the value of a policy. Finally, although we do not have direct access to the mentor's policy π_E, we can compute a good estimate of $\boldsymbol{\mu}_E$ when the number of trajectories demonstrated by the mentor is large.

None of the methods described above produces an exact result, so to complete the analysis, we must explain how the approximation errors introduced by these methods will affect the performance guarantees in Corollaries 4.2 and 4.4. However, to simplify

our presentation, we defer these issues until Section 5.3. In that section, we provide a unified analysis of the MWAL algorithm — an apprenticeship learning algorithm that is the main topic of Chapter 5 — that takes these approximation errors into account. In that section, we prove that the performance of the MWAL algorithm degrades gracefully as the approximation error increases. That unified analysis applies, with only minor changes, to both the Projection algorithm and the Blackwell algorithm.

4.5 Reduction to Classification

The apprenticeship learning algorithms described in the previous section assume that the true reward function can be written as a linear combination of a set of known features. However, there may be cases where the apprentice is unwilling or unable to assume that the rewards have this structure. In such a scenario, is there any way for the apprentice to take advantage of demonstrations provided by the mentor? In this section, we show how apprenticeship learning can be *reduced* to classification. In other words, we explain how an apprentice can use a classification algorithm to mimic the mentor's behavior, and how the error of the learned classifier bounds the difference between the value of the apprentice's policy and the value of the mentor's policy. The idea of reducing one learning problem to another was first proposed by Zadrozny et al. [147].

The assumptions made in this section are slightly different than in the previous section, and we explain these differences in Section 4.5.1. We provide the details of the reduction in Section 4.5.2, and in Section 4.5.3 we prove our first guarantee about this reduction: The difference between the value of the apprentice's policy and the mentor's policy is $O(\sqrt{\epsilon})$, where $\epsilon \in [0, 1]$ is the error of the learned classifier. In Section 4.5.4, we prove that this difference is only $O(\epsilon)$ when the mentor's policy is close to optimal.

4.5.1 Preliminaries

In this section, we will allow the state space $\mathcal{S}$ to be infinite, but will continue to assume that the action space $\mathcal{A}$ is finite. Thus the initial state distribution $\alpha(\cdot)$ specifies a distribution on a possibly infinite set of states, as does the transition function $\theta(s, a, \cdot)$ for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$. Further, the only assumptions we make about the reward function R is that $0 \leq R(s) \leq R^{\max}$ for all states $s \in \mathcal{S}$, where $R^{\max}$ is a finite upper bound on the reward of any state.

Unlike Section 4.2, in this section we will study *nonstationary* policies in a *finite-horizon* MDP, with horizon length H. Let Π be the set of all stationary policies. A policy π is *nonstationary* if it belongs to the set $\Pi^H = \Pi \times \cdots (H \text{ times}) \cdots \times \Pi$. In this case, $\pi_t(s, a)$ denotes the probability of taking action a in state s at time t. Also, if π is nonstationary, then π_t refers to the stationary policy that is equal to the t^{th} component of π.

A (stationary or nonstationary) policy π is *deterministic* if each one of its action distributions is concentrated on a single action. If a deterministic policy π is stationary, then $\pi(s)$ is the action taken in state s, and if π is nonstationary, the $\pi_t(s)$ is the action taken in state s at time t.

Much as we did for stationary policies in an infinite-horizon MDP, we define the value function $V_t^\pi(s)$ for a nonstationary policy π at time t as follows:

$$V_t^\pi(s) \triangleq E\left[\sum_{t'=t}^{H} R(s_{t'}) \;\middle|\; s_t = s, a_{t'} \sim \pi_{t'}(s_{t'}, \cdot), s_{t'+1} \sim \theta(s_{t'}, a_{t'}, \cdot)\right]$$

So $V_t^\pi(s)$ is the expected cumulative reward for following policy π when starting at state s and time step t. Note that, unlike stationary policies, there are several value functions per nonstationary policy, one for each time step t. The value of a policy is defined to be

$$V(\pi) \triangleq E[V_1^\pi(s) \mid s \sim \alpha(\cdot)]$$

and an optimal policy π^* is one that satisfies $\pi^* \triangleq \arg\max_\pi V(\pi)$. The existence of this maximum is a consequence of the compactness of $\Delta^{|\mathcal{A}|}$ (the set of distributions on $|\mathcal{A}|$ elements) and the continuity of $V(\pi)$.

We write π^E to denote the mentor's policy,[4] and $V_t^E(s)$ as an abbreviation for $V_t^{\pi^E}(s)$.

Let D_t^π be the distribution on state-action pairs at time t under policy π. In other words, a sample (s, a) is drawn from D_t^π by first drawing $s_1 \sim \alpha(\cdot)$, then following policy π for time steps 0 through t, which generates a trajectory $(s_1, a_1, \ldots, s_t, a_t)$, and then letting $(s, a) = (s_t, a_t)$. We write D_t^E as an abbreviation for $D_t^{\pi^E}$. In a minor abuse of notation, we write $s \sim D_t^\pi$ to mean: draw state-action pair $(s, a) \sim D_t^\pi$, and discard a.

4.5.2 Details of the Reduction

Our goal is to reduce apprenticeship learning to classification, so let us describe exactly how this reduction is defined.

In a classification problem, a learning algorithm is given a training set $(x_1, y_1, \ldots, x_m, y_m)$, where each labeled example $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ is drawn independently from a distribution D on $\mathcal{X} \times \mathcal{Y}$. Here $\mathcal{X}$ is the example space and $\mathcal{Y}$ is the finite set of labels. The learning algorithm is also given the definition of a hypothesis class $\mathcal{H}$, which is a set of functions mapping $\mathcal{X}$ to $\mathcal{Y}$. The objective of the learning algorithm is to find a hypothesis $h \in \mathcal{H}$ such that the error $\Pr_{(x,y)\sim D}(h(x) \neq y)$ is small.

The hypothesis class $\mathcal{H}$ is said to be *PAC-learnable* (where "PAC" stands for "Probably Approximately Correct") if there exists a learning algorithm A such that, whenever A is given a training set of size $m = \text{poly}(\frac{1}{\delta}, \frac{1}{\epsilon})$, the algorithm runs for $\text{poly}(\frac{1}{\delta}, \frac{1}{\epsilon})$ steps and outputs a hypothesis $\hat{h} \in \mathcal{H}$ such that, with probability at least

[4]Note that, in constrast to Section 4.3, we have moved the 'E' in π^E from the subscript to the superscript, to make room for a time index

$1 - \delta$

$$\Pr_{(x,y)\sim D}\left(\hat{h}(x) \neq y\right) \leq \epsilon^*_{\mathcal{H},D} + \epsilon$$

Here $\epsilon^*_{\mathcal{H},D} = \inf_{h\in\mathcal{H}} \Pr_{(x,y)\sim D}(h(x) \neq y)$ is the error of the best hypothesis in $\mathcal{H}$. The expression $\text{poly}(\frac{1}{\delta}, \frac{1}{\epsilon})$ will typically also depend on other quantities, such as the number of labels $|\mathcal{Y}|$ and the *VC-dimension* of $\mathcal{H}$ [135], but this dependence is not germane to our discussion. PAC learnability is one of the oldest and most well-studied frameworks for machine learning, and PAC learning algorithms have been developed for many hypothesis classes [57].[5]

The existence of PAC-learnable hypothesis classes will allow us to reduce the apprenticeship learning problem to classification. Suppose that the apprentice observes m independent trajectories from the mentor's policy π^E, where the ith trajectory is a sequence $(s^i_1, a^i_1, \ldots, s^i_H, a^i_H)$. The key is to note that each (s^i_t, a^i_t) can be viewed as an independent sample from the distribution D^E_t. Now consider a PAC-learnable hypothesis class $\mathcal{H}$, where $\mathcal{H}$ contains a set of functions mapping the state space $\mathcal{S}$ to the finite action space $\mathcal{A}$. If $m = \text{poly}(\frac{1}{H\delta}, \frac{1}{\epsilon})$, then for each time step t, the apprentice can use a PAC learning algorithm for $\mathcal{H}$ to learn a hypothesis $\hat{h}_t \in \mathcal{H}$ such that, with probability at least $1 - \frac{1}{H\delta}$

$$\Pr_{(s,a)\sim D^E_t}\left(\hat{h}_t(s) \neq a\right) \leq \epsilon^*_{\mathcal{H},D^E_t} + \epsilon$$

And by the union bound, this inequality holds for *all* t with probability at least $1-\delta$. If each $\epsilon^*_{\mathcal{H},D^E_t} + \epsilon$ is small, then a natural choice for the apprentice's policy π^A is to set $\pi^A_t = \hat{h}_t$ for all t. This policy uses the learned classifiers to mimic the behavior of the mentor.

In light of the preceding discussion, throughout the remainder of this section we make the following assumption about the apprentice's policy.

[5]Technically, the definition given here is for *agnostic* PAC-learnability [45, 58], a somewhat more advanced concept that the original definition of PAC-learnability.

Assumption 4.5. *The apprentice policy π^A is a deterministic policy that satisfies*

$$\Pr_{(s,a)\sim D_t^E}(\pi_t^A(s) \neq a) \leq \epsilon$$

for some $\epsilon > 0$ and all time steps t.

As we have shown, an apprentice policy satisfying Assumption 4.5 with small ϵ can be found with high probability, provided that mentor's policy is well-approximated by a PAC-learnable hypothesis class and that the apprentice is given enough trajectories from the mentor. A reasonable intuition is that the value of the policy π^A in Assumption 4.5 is nearly as high as the value of the policy π^E; the remainder of this section is devoted to confirming this intuition.

4.5.3 Guarantee for Any Mentor

If the error rate ϵ in Assumption 4.5 is small, then the apprentice's policy π^A closely mimics the mentor's policy π^E, and we might hope that this implies that $V(\pi^A)$ is not much less than $V(\pi^E)$. This is indeed the case, as the next theorem shows.

Theorem 4.6. *If Assumption 4.5 holds, then*

$$V(\pi^A) \geq V(\pi^E) - 2\sqrt{\epsilon}H^2R^{\max}$$

In a typical classification problem, it is assumed that the training and test examples are drawn from the same distribution. The main challenge in proving Theorem 4.6 is that this assumption does *not* hold for the classification problems to which we have reduced the apprenticeship learning problem. This is because, although each state-action pair (s_t^i, a_t^i) appearing in a mentor trajectory is distributed according to D_t^E, a state-action pair (s_t, a_t) visited by the apprentice's policy may not follow this distribution, since the behavior of the apprentice prior to time step t may not exactly

match the mentor's behavior. So our strategy for proving Theorem 4.6 will be to show that these differences do not cause the value of the apprentice policy to degrade too much relative to the value of the mentor's policy.

Before proceeding, we will show that Assumption 4.5 implies a condition that is, for our purposes, more convenient.

Lemma 4.7. *Let $\hat{\pi}$ be a deterministic nonstationary policy. If*

$$\Pr_{(s,a)\sim D_t^E}(\hat{\pi}_t(s) \neq a) \leq \epsilon$$

then for all $\epsilon_1 \in [0,1]$

$$\Pr_{s\sim D_t^E}\left(\pi_t^E(s, \hat{\pi}_t(s)) \geq 1 - \epsilon_1\right) \geq 1 - \tfrac{\epsilon}{\epsilon_1}$$

Proof. Fix any $\epsilon_1 \in [0,1]$, and suppose for contradiction that

$$\Pr_{s\sim D_t^E}\left(\pi_t^E(s, \hat{\pi}_t(s)) \geq 1 - \epsilon_1\right) < 1 - \tfrac{\epsilon}{\epsilon_1}$$

Say that a state s is *good* if $\pi_t^E(s, \hat{\pi}_t(s)) \geq 1 - \epsilon_1$, and that s is *bad* otherwise. Then

$$\begin{aligned}
\Pr_{(s,a)\sim D_t^E}(\hat{\pi}_t(s) = a) &= \Pr_{s\sim D_t^E}(s \text{ is good}) \cdot \Pr_{(s,a)\sim D_t^E}(\hat{\pi}_t(s) = a \mid s \text{ is good}) \\
&\quad + \Pr_{s\sim D_t^E}(s \text{ is bad}) \cdot \Pr_{(s,a)\sim D_t^E}(\hat{\pi}_t(s) = a \mid s \text{ is bad}) \\
&< \Pr_{s\sim D_t^E}(s \text{ is good}) \cdot 1 + (1 - \Pr_{s\sim D_t^E}(s \text{ is good})) \cdot (1 - \epsilon_1) \\
&= 1 - \epsilon_1(1 - \Pr_{s\sim D_t^E}(s \text{ is good})) \\
&< 1 - \epsilon
\end{aligned}$$

where the first inequality holds because $\Pr_{(s,a)\sim D_t^E}(\hat{\pi}_t(s) = a \mid s \text{ is bad}) < 1 - \epsilon_1$, and the second inequality holds because $\Pr_{s\sim D_t^E}(s \text{ is good}) < 1 - \frac{\epsilon}{\epsilon_1}$. This chain of inequalities clearly contradicts the assumption of the lemma.

□

The next two lemmas are the main tools used to prove Theorem 4.6. In the proofs of these lemmas, we write $\overline{sa}$ to denote a trajectory, where $\overline{sa} = (\bar{s}_1, \bar{a}_1, \ldots, \bar{s}_H, \bar{a}_H) \in (\mathcal{S} \times \mathcal{A})^H$. Also, let dP_π denote the probabilty measure induced on trajectories by following policy π, and let $R(\overline{sa}) = \sum_{t=1}^{H} R(\bar{s}_t)$ denote the sum of the rewards of the states in trajectory $\overline{sa}$. Importantly, using these definitions we have

$$V(\pi) = \int_{\overline{sa}} R(\overline{sa}) dP_\pi$$

The next lemma proves that if a deterministic policy "almost" agrees with the mentor's policy π^E in every state and time step, then its value is not much worse the the value of π^E.

Lemma 4.8. *Let $\hat{\pi}$ be a deterministic nonstationary policy. If for all states s and time steps t*

$$\pi_t^E(s, \hat{\pi}_t(s)) \geq 1 - \epsilon$$

then

$$V(\hat{\pi}) \geq V(\pi^E) - \epsilon H^2 R^{\max}$$

Proof. Say a trajectory $\overline{sa}$ is *good* if it is "consistent" with $\hat{\pi}$ — that is, $\hat{\pi}(\bar{s}_t) = \bar{a}_t$

for all time steps t — and that $\overline{sa}$ is *bad* otherwise. We have

$$\begin{aligned}
V(\pi^E) &= E\left[\sum_{t=1}^{H} R(s_t) \;\middle|\; s_1 \sim \alpha(\cdot), a_t \sim \pi_t^E(s_t, \cdot), s_{t+1} \sim \theta(s_t, a_t, \cdot)\right] \\
&= \int_{\overline{sa}} R(\overline{sa}) dP_{\pi^E} \\
&= \int_{\overline{sa}\text{ good}} R(\overline{sa}) dP_{\pi^E} + \int_{\overline{sa}\text{ bad}} R(\overline{sa}) dP_{\pi^E} \\
&\leq \int_{\overline{sa}\text{ good}} R(\overline{sa}) dP_{\pi^E} + \epsilon H^2 R^{\max} \\
&\leq \int_{\overline{sa}\text{ good}} R(\overline{sa}) dP_{\hat{\pi}} + \epsilon H^2 R^{\max} \\
&= V(\hat{\pi}) + \epsilon H^2 R^{\max}
\end{aligned}$$

where the first inequality holds because, by the union bound, P_{π^E} assigns at most an ϵH fraction of its measure to bad trajectories, and the maximum reward of a trajectory is $HR^{\max}$. The second inequality holds because good trajectories are assigned at least as much measure by $P_{\hat{\pi}}$ as by P_{π^E}, because $\hat{\pi}$ is deterministic. □

The next lemma proves a slightly different statement than Lemma 4.8: If a policy exactly agrees with the mentor's policy π^E in "almost" every state and time step, then its value is not much worse the the value of π^E.

Lemma 4.9. *Let $\hat{\pi}$ be a nonstationary policy. If for all time steps t*

$$\Pr_{s \sim D_t^E}\left(\hat{\pi}_t(s, \cdot) = \pi_t^E(s, \cdot)\right) \geq 1 - \epsilon$$

then

$$V(\hat{\pi}) \geq V(\pi^E) - \epsilon H^2 R^{\max}$$

Proof. Say a trajectory $\overline{sa}$ is *good* if $\pi_t^E(\bar{s}_t, \cdot) = \hat{\pi}_t(\bar{s}_t, \cdot)$ for all time steps t, and that

$\overline{sa}$ is *bad* otherwise. We have

$$\begin{aligned}
V(\hat{\pi}) &= E\left[\sum_{t=1}^{H} R(s_t) \;\middle|\; s_1 \sim \alpha(\cdot), a_t \sim \hat{\pi}_t(s_t, \cdot), s_{t+1} \sim \theta(s_t, a_t, \cdot)\right] \\
&= \int_{\overline{sa}} R(\overline{sa}) dP_{\hat{\pi}} \\
&= \int_{\overline{sa}\text{ good}} R(\overline{sa}) dP_{\hat{\pi}} + \int_{\overline{sa}\text{ bad}} R(\overline{sa}) dP_{\hat{\pi}} \\
&= \int_{\overline{sa}\text{ good}} R(\overline{sa}) dP_{\pi^E} + \int_{\overline{sa}\text{ bad}} R(\overline{sa}) dP_{\hat{\pi}} \\
&= \int_{\overline{sa}} R(\overline{sa}) dP_{\pi^E} - \int_{\overline{sa}\text{ bad}} R(\overline{sa}) dP_{\pi^E} + \int_{\overline{sa}\text{ bad}} R(\overline{sa}) dP_{\hat{\pi}} \\
&\geq V(\pi^E) - \epsilon H^2 R^{\max} + \int_{\overline{sa}\text{ bad}} R(\overline{sa}) dP_{\hat{\pi}} \\
&\geq V(\pi^E) - \epsilon H^2 R^{\max}
\end{aligned}$$

The first inequality holds because, by the union bound, P_{π^E} assigns at most an ϵH fraction of its measure to bad trajectories, and the maximum reward of a trajectory is $HR^{\max}$. The second inequality holds by our assumption that all rewards are nonnegative. □

We are now ready to combine the previous lemmas and prove Theorem 4.6.

Proof of Theorem 4.6. Since the apprentice's policy π^A satisfies Assumption 4.5, by Lemma 4.7 we can chose any $\epsilon_1 \in [0, 1]$ and have

$$\Pr_{s \sim D_t^E}\left(\pi_t^E(s, \pi_t^A(s)) \geq 1 - \epsilon_1\right) \geq 1 - \tfrac{\epsilon}{\epsilon_1}$$

Now construct a "dummy" policy $\hat{\pi}$ as follows: For all time steps t, let $\hat{\pi}_t(s, \cdot) = \pi_t^E(s, \cdot)$ for any state s where $\pi_t^E(s, \pi_t^A(s)) \geq 1 - \epsilon_1$. On all other states, let $\hat{\pi}_t(s, \pi_t^A(s)) = 1$. By Lemma 4.8

$$V(\pi^A) \geq V(\hat{\pi}) - \epsilon_1 H^2 R^{\max}$$

and by Lemma 4.9

$$V(\hat{\pi}) \geq V(\pi^E) - \frac{\epsilon}{\epsilon_1} H^2 R^{\max}$$

Combining these inequalities yields

$$V(\pi^A) \geq V(\pi^E) - \left(\epsilon_1 + \frac{\epsilon}{\epsilon_1}\right) H^2 R^{\max}$$

Since ϵ_1 was chosen arbitrarily, we set $\epsilon_1 = \sqrt{\epsilon}$, which maximizes this lower bound. □

4.5.4 Guarantee for Good Mentor

Theorem 4.6 makes no assumptions about the value of the mentor's policy. However, in many cases it may be reasonable to assume that the mentor is following a near-optimal policy (indeed, if she is not, then we should question the decision to select her as mentor). The next theorem shows that the dependence of $V(\pi^A)$ on the classification error ϵ is significantly better when the mentor is following a near-optimal policy.

Theorem 4.10. *If Assumption 4.5 holds, then*

$$V(\pi^A) \geq V(\pi^E) - \left(4\epsilon H^3 R^{\max} + \Delta_{\pi^E}\right)$$

where $\Delta_{\pi^E} \triangleq V(\pi^*) - V(\pi^E)$ *is the* suboptimality *of the mentor's policy* π^E.

Note that, when $\Delta_{\pi^E} \leq O\left(\epsilon H^3 R^{\max}\right)$, the bound in Theorem 4.10 varies with ϵ and not with $\sqrt{\epsilon}$. To see why a near-optimal mentor policy should yield a weaker dependence on ϵ for our reduction, consider a mentor policy π^E that is an optimal policy, but in every state $s \in \mathcal{S}$ selects one of two actions a_1^s and a_2^s uniformly at random. A deterministic apprentice policy π^A that closely mimics the mentor will either set $\pi^A(s) = a_1^s$ or $\pi^A(s) = a_2^s$, but in either case the classification error will not

be less than $\frac{1}{2}$. However, since π^E is optimal, both actions a_1^s and a_2^s must be optimal actions for state s, and so the apprentice policy π^A will be optimal as well.

Our strategy for proving Theorem 4.10 is to replace Lemma 4.8 with a different result — namely, Lemma 4.13 below — that has a much weaker dependence on the classification error ϵ when Δ_{π^E} is small.

To help us prove Lemma 4.13, we will first need to define several useful policies. The next several definitions will be with respect to an arbitrary nonstationary *base policy* π^B; in the proof of Theorem 4.10, we will make a particular choice for the base policy.

Fix a deterministic nonstationary policy $\pi^{B,\epsilon}$ that satisfies

$$\pi_t^B(s, \pi_t^{B,\epsilon}(s)) \geq 1 - \epsilon$$

for some $\epsilon \in [0, 1]$ and all states s and time steps t. Such a policy always exists by letting $\epsilon = 1$, but if ϵ is close to zero, then $\pi^{B,\epsilon}$ is a deterministic policy that "almost" agrees with π^B in every state and time step. Of course, depending on the choice of π^B, a policy $\pi^{B,\epsilon}$ may not exist for small ϵ, but let us set aside that concern for the moment; in the proof of Theorem 4.10, the base policy π^B will chosen so that ϵ can be as small as we like.

Having thusly defined $\pi^{B,\epsilon}$, we define $\pi^{B\setminus\epsilon}$ as follows: For all states $s \in \mathcal{S}$ and time steps t, if $\pi_t^B(s, \pi^{B,\epsilon}(s)) < 1$, then let

$$\pi_t^{B\setminus\epsilon}(s, a) = \begin{cases} 0 & \text{if } \pi_t^{B,\epsilon}(s) = a \\ \dfrac{\pi_t^B(s, a)}{\sum_{a' \neq \pi_t^{B,\epsilon}(s)} \pi_t^B(s, a')} & \text{otherwise} \end{cases}$$

for all actions $a \in \mathcal{A}$, and otherwise let $\pi_t^{B\setminus\epsilon}(s, a) = \frac{1}{|\mathcal{A}|}$ for all $a \in \mathcal{A}$. In other words, in each state s and time step t, the distribution $\pi_t^{B\setminus\epsilon}(s, \cdot)$ is obtained by proportionally

redistributing the probability assigned to action $\pi_t^{B,\epsilon}(s)$ by the distribution $\pi_t^B(s,\cdot)$ to all other actions. The case where $\pi_t^B(s,\cdot)$ assigns all probability to action $\pi_t^{B,\epsilon}(s)$ is treated specially, but as will be clear from the proof of Lemma 4.11, it is actually immaterial how the distribution $\pi_t^{B\setminus\epsilon}(s,\cdot)$ is defined in these cases; we choose the uniform distribution for definiteness.

Let π^{B+} be a deterministic policy defined by

$$\pi_t^{B+}(s) = \arg\max_a E\left[V_{t+1}^{\pi^B}(s') \;\middle|\; s' \sim \theta(s,a,\cdot)\right]$$

for all states $s \in \mathcal{S}$ and time steps t. In other words, $\pi_t^{B+}(s)$ is the best action in state s at time t, assuming that the policy π^B is followed thereafter.

The next definition requires the use of mixed policies. Fundamentally, our definition of a mixed policy in this section will be the same as it was in Section 4.3: A mixed policy consists of a finite set of deterministic policies, along with a distribution over those policies; the mixed policy is followed by drawing a single policy according to the distribution in the initial time step, and following that policy exclusively thereafter. However, our formal definition will require somewhat different notational syntax, since in this section the set of all deterministic policies may not be finite, or even countable, so we cannot assign a unique index to every deterministic policy.

Formally, a mixed policy is defined by a set of ordered pairs $\{(\pi^i, \lambda(i))\}_{i=1}^N$ for some finite N, where each *component policy* π^i is a deterministic nonstationary policy, $\sum_{i=1}^N \lambda(i) = 1$ and $\lambda(i) \geq 0$ for all $i \in [N]$.

Let $\tilde{\pi}^{B,\epsilon,+}$ be a mixed policy consisting of $N = 2^{|H|}$ component policies. It is easiest to describe the distribution assigned by $\tilde{\pi}^{B,\epsilon,+}$ to these component policies in terms of a generative procedure. Each component policy π^i is drawn from the

distribution as follows:

$$\pi_t^i = \begin{cases} \pi_t^{B,\epsilon} & \text{with probability } (1-\epsilon) \\ \pi_t^{B+} & \text{with probability } \epsilon \end{cases}$$

for all time steps t. So the probability $\lambda(i)$ assigned to each component policy π^i in the mixed policy is $\lambda(i) = (1-\epsilon)^{k(i)}\epsilon^{H-k(i)}$, where $k(i)$ is the number of times steps t for which $\pi_t^i = \pi_t^{B,\epsilon}$.

Having established these definitions, we are now ready to prove several lemmas that will help us prove Theorem 4.10.

Lemma 4.11. $V(\tilde{\pi}^{B,\epsilon,+}) \geq V(\pi^B)$

Proof. The proof will be by induction. Clearly $V_H^{\tilde{\pi}^{B,\epsilon,+}}(s) = V_H^{\pi^B}(s)$ for all states s, since the value function V_H^π for any policy π depends only on the reward function R. Now suppose for induction that $V_{t+1}^{\tilde{\pi}^{B,\epsilon,+}}(s) \geq V_{t+1}^{\pi^B}(s)$ for all states s. Then for all states s

$$\begin{aligned}
V_t^{\tilde{\pi}^{B,\epsilon,+}}(s) &= R(s) + E\left[V_{t+1}^{\tilde{\pi}^{B,\epsilon,+}}(s') \;\middle|\; a' \sim \tilde{\pi}_t^{B,\epsilon,+}(s,\cdot), s' \sim \theta(s,a',\cdot)\right] \\
&\geq R(s) + E\left[V_{t+1}^{\pi^B}(s') \;\middle|\; a' \sim \tilde{\pi}_t^{B,\epsilon,+}(s,\cdot), s' \sim \theta(s,a',\cdot)\right] \\
&= R(s) + (1-\epsilon)E\left[V_{t+1}^{\pi^B}(s') \;\middle|\; s' \sim \theta(s,\pi_t^{B,\epsilon}(s),\cdot)\right] + \epsilon E\left[V_{t+1}^{\pi^B}(s') \;\middle|\; s' \sim \theta(s,\pi_t^{B+}(s),\cdot)\right] \\
&\geq R(s) + \pi_t^B(s,\pi_t^{B,\epsilon}(s)) \cdot E\left[V_{t+1}^{\pi^B}(s') \;\middle|\; s' \sim \theta(s,\pi_t^{B,\epsilon}(s),\cdot)\right] \\
&\qquad + \left(1 - \pi_t^B(s,\pi_t^{B,\epsilon}(s))\right) \cdot E\left[V_{t+1}^{\pi^B}(s') \;\middle|\; s' \sim \theta(s,\pi_t^{B+}(s),\cdot)\right] \\
&\geq R(s) + \pi_t^B(s,\pi_t^{B,\epsilon}(s)) \cdot E\left[V_{t+1}^{\pi^B}(s') \;\middle|\; s' \sim \theta(s,\pi_t^{B,\epsilon}(s),\cdot)\right] \\
&\qquad + \left(1 - \pi_t^B(s,\pi_t^{B,\epsilon}(s))\right) \cdot E\left[V_{t+1}^{\pi^B}(s') \;\middle|\; a' \sim \pi_t^{B\setminus\epsilon}(s,\cdot), s' \sim \theta(s,a',\cdot)\right] \\
&= R(s) + E\left[V_{t+1}^{\pi^B}(s') \;\middle|\; a' \sim \pi_t^B(s), s' \sim \theta(s,a',\cdot)\right] \\
&= V_t^{\pi^B}(s)
\end{aligned}$$

The first equality holds for all policies π, and follows straightforwardly from the definition of V_t^π. The rest of the derivation uses, in order: the inductive hypothesis; the definition of $\tilde{\pi}^{B,\epsilon,+}$; property of $\pi^{B,\epsilon}$ and fact that $\pi_t^{B+}(s)$ is the best action with respect to $V_{t+1}^{\pi^B}$; the fact that $\pi_t^{B+}(s)$ is the best action with respect to $V_{t+1}^{\pi^B}$; the definition of $\pi^{B\setminus\epsilon}$; the definition of $V_t^{\pi^B}(s)$. □

Lemma 4.12. $V(\tilde{\pi}^{B,\epsilon,+}) \leq (1-\epsilon H)V(\pi^{B,\epsilon}) + \epsilon H V(\pi^*)$

Proof. Since $\tilde{\pi}^{B,\epsilon,+}$ is a mixed policy, by the linearity of expectation we have

$$V(\tilde{\pi}^{B,\epsilon,+}) = \sum_{i=1}^{N} \lambda(i) V(\pi^i)$$

where each π^i is a component policy of $\tilde{\pi}^{B,\epsilon,+}$ and $\lambda(i)$ is its associated probability. Therefore

$$\begin{aligned} V(\tilde{\pi}^{B,\epsilon,+}) &= \sum_i \lambda(i) V(\pi^i) \\ &\leq (1-\epsilon)^H V(\pi^{B,\epsilon}) + (1-(1-\epsilon)^H) V(\pi^*) \\ &\leq (1-\epsilon H) V(\pi^{B,\epsilon}) + \epsilon H V(\pi^*) \end{aligned}$$

Here we used the fact that probability $(1-\epsilon)^H \geq 1-\epsilon H$ is assigned to a component policy that is identical to $\pi^{B,\epsilon}$, and the value of any component policy is at most $V(\pi^*)$. □

Lemma 4.13. *If* $\epsilon < \frac{1}{H}$, *then* $V(\pi^{B,\epsilon}) \geq V(\pi^B) - \frac{\epsilon H}{1-\epsilon H}\Delta_{\pi^B}$

Proof. Combining Lemmas 4.11 and 4.12 yields

$$(1-\epsilon H)V(\pi^{B,\epsilon}) + \epsilon H V(\pi^*) \geq V(\pi^B)$$

And via algebraic manipulation we have

$$\begin{aligned}
&(1-\epsilon H)V(\pi^{B,\epsilon}) + \epsilon H V(\pi^*) \geq V(\pi^B) \\
\Rightarrow\ &(1-\epsilon H)V(\pi^{B,\epsilon}) \geq (1-\epsilon H)V(\pi^B) + \epsilon H V(\pi^B) - \epsilon H V(\pi^*) \\
\Rightarrow\ &(1-\epsilon H)V(\pi^{B,\epsilon}) \geq (1-\epsilon H)V(\pi^B) - \epsilon H \Delta_{\pi^B} \\
\Rightarrow\ &V(\pi^{B,\epsilon}) \geq V(\pi^B) - \frac{\epsilon H}{1-\epsilon H}\Delta_{\pi^B}
\end{aligned}$$

In the last line, we were able to divide by $(1-\epsilon H)$ without changing the direction of the inequality because of our assumption that $\epsilon < \frac{1}{H}$. □

We are now ready to combine the previous lemmas and prove Theorem 4.10.

Proof of Theorem 4.10. Since the apprentice's policy π^A satisfies Assumption 4.5, by Lemma 4.7 we can chose any $\epsilon_1 \in [0, \frac{1}{H})$ and have

$$\Pr_{s\sim D_t^E}\left(\pi_t^E(s, \pi_t^A(s)) \geq 1-\epsilon_1\right) \geq 1 - \frac{\epsilon}{\epsilon_1}$$

As in the proof of Theorem 4.6, let us construct a "dummy" policy $\hat{\pi}$ as follows: For all time steps t, let $\hat{\pi}_t(s,\cdot) = \pi_t^E(s,\cdot)$ for any state s where $\pi_t^E(s, \pi_t^A(s)) \geq 1-\epsilon_1$. On all other states, let $\hat{\pi}_t(s, \pi_t^A(s)) = 1$. By Lemma 4.9 we have

$$V(\hat{\pi}) \geq V(\pi^E) - \frac{\epsilon}{\epsilon_1} H^2 R^{\max} \tag{4.14}$$

Substituting $V(\pi^E) = V(\pi^*) - \Delta_{\pi^E}$ and $V(\hat{\pi}) = V(\pi^*) - \Delta_{\hat{\pi}}$ and rearranging yields

$$\Delta_{\hat{\pi}} \leq \Delta_{\pi^E} + \frac{\epsilon}{\epsilon_1} H^2 R^{\max} \tag{4.15}$$

Now observe that, if we set the base policy $\pi^B = \hat{\pi}$, then by definition π^A is a valid

choice for π^{B,ϵ_1}. And since $\epsilon_1 < \frac{1}{H}$ we have

$$\begin{aligned} V(\pi^A) &\geq V(\hat{\pi}) - \frac{\epsilon_1 H}{1-\epsilon_1 H}\Delta_{\hat{\pi}} \\ &\geq V(\hat{\pi}) - \frac{\epsilon_1 H}{1-\epsilon_1 H}\left(\Delta_{\pi^E} + \frac{\epsilon}{\epsilon_1}H^2 R^{\max}\right) \\ &\geq V(\pi^E) - \frac{\epsilon}{\epsilon_1}H^2 R^{\max} - \frac{\epsilon_1 H}{1-\epsilon_1 H}\left(\Delta_{\pi^E} + \frac{\epsilon}{\epsilon_1}H^2 R^{\max}\right) \end{aligned} \tag{4.16}$$

where we used Lemma 4.13, (4.15) and (4.14), in that order. Letting $\epsilon_1 = \frac{1}{2H}$ proves the theorem. □

4.6 Other Related Work

Learning behavior from a mentor has a long history in machine learning. Some of the earliest and most influential work was by Sammut et al. [112], who studied the problem of learning to pilot a flight simulator. However, the idea of mimicking mentor behavior via a reward function was relatively unexplored prior to the introduction of the apprenticeship learning framework by Abbeel and Ng [1]. However Atkeson and Schaal [4], who considered the problem of having a robot arm follow a demonstrated trajectory, represent a notable exception. Their algorithm used a reward function to penalize deviations from the trajectory.

The apprenticeship learning framework is closely related to *inverse reinforcement learning*, first proposed by Ng and Russell [89]. In the inverse reinforcement learning problem, the objective is to learn a reward function for which an observed policy is optimal. Note that recovering the reward function is the explicit goal of this approach, unlike apprenticeship learning, where the true reward function need not be learned.

In some cases, attempting to recover the true reward function *is* an effective method for learning a good apprentice policy, particularly when one makes the additional assumption that the mentor policy is optimal. For example, the goal of

Neu and Szepesvari [88] was to learn a reward function for which an approximately optimal policy with respect to that reward function approximately mimics the mentor. They formulated their problem as nondifferentiable optimization, and solved the optimization via a subgradient method.

Similarly, in *max margin planning* [105, 104], the goal is to learn a reward function so that, with respect to this reward funciton, the demonstrated policy is nearly better than all other policies. The magnitude of this advantage over each comparison policy, also known as the margin, scales with the loss of the policy, which is usually defined as a measure of how different it is from the demonstrated policy. The idea of maximizing a margin over a set of complex objects, such as policies, is the basis of *structured prediction*, first introduced by Taskar et al. [130]; see also Tsochantaridis et al. [132].

Most apprenticeship learning algorithms assume that it is easy for a mentor to provide complete trajectories demonstrating the desired behavior. However, Kolter et al. [64] studied a setting where it is only feasible for a mentor to provide partial trajectories. In particular, they studied a quadraped locomotion task, in which a mentor is only able to provide advice at two hierarchical levels (an overall plan for moving through an obstacle course, and how to navigate around individual obstacles). These partial trajectories are used to learn policies at each hierarchical level, which are then combined into a single policy.

The Blackwell algorithm, presented in Section 4.4.2, has some similarity to the algorithm introduced by Mannor and Shimkin [75]. The main differences are that Mannor and Shimkin [75] did not specifically study the apprenticeship learning problem, and, more significantly, their algorithm is designed for a setting where the agent is maximizing expected *average* reward, not expected cumulative discounted reward.

Several authors have reduced reinforcement learning to a simpler problem, much as we reduced apprenticeship learning to classification in Section 4.5. Bagnell et al. [5] described an algorithm for constructing a good nonstationary policy from a sequence

of good "one-step" policies. These policies are only concerned with maximizing reward collected in a single time step, and are learned with the help of observations from a mentor. Langford and Zadrozny [66] reduced reinforcement learning to sequence of classification problems (see also Blatt and Hero [11]), but these problems have an unusual structure, and Langford and Zadrozny [66] provide no guidance as to how data for these problems can be collected. Kakade and Langford [52] reduced reinforcement learning to regression, but required additional assumptions about how easily a learning algorithm can access the entire state space. All this work assumes that the true rewards are known.

Other authors have focused on empirical evaluations of using classifiers in reinforcement learning problems, such as Rexakis and Lagoudakis [106]. Relatedly, Lagoudakis and Parr [65] describe a method for using a classifier in the policy evaluation step of a policy iteration algorithm.

4.7 Conclusion

In this chapter, we reviewed both the reinforcement learning and apprenticeship learning frameworks, and described the most widely-used algorithm for apprenticeship learning. We presented two new apprenticeship learning algorithms, one of which is both simpler and faster than previous methods, and another which requires much less information about the true rewards.

Chapter 5

Game-Theoretic Approach to Apprenticeship Learning

The apprenticeship learning algorithms in Chapter 4 are all based on mimicking the mentor. In this chapter, we describe algorithms that take a very different approach. We reformulate the apprenticeship learning framework as a two-player zero-sum game. The apprentice is a player in this game, and her strategies are all the possible policies. The other player represents an adversary whose strategies are all possible choices of reward functions. The goal of the apprentice is to choose a policy that has maximum value relative to the mentor, assuming that the reward function will be adversarially selected with respect to this goal. Since the true reward function is unknown, we argue that this kind of worst-case assumption is appropriate.

We provide a detailed overview of our game-theoretic framework in Section 5.1. In our formulation, the apprentice has an extremely large number of strategies in the game, far too many to explicitly enumerate. As we explained in Chapters 2 and 3, no-regret algorithms such as the MW algorithm are especially well-suited to solving zero-sum games in which one player has a large number of strategies, provided that one has access to an efficient best-response oracle for that player. Fortunately, the

required best-response oracles can be efficiently implemented using the classical solution methods for reinforcement learning problems that we reviewed in Section 4.2.1. Thus, we can adapt the MW algorithm to find a good apprentice policy by using one of these methods as a subroutine. A strength of our approach is its modularity; since the MW algorithm can use essentially any existing reinforcement learning algorithm as a best-response oracle, it inherits all the advantages of modern algorithms, such as the ability to handle large environments. We present the Multiplicative Weights for Apprenticeship Learning (MWAL) algorithm in Section 5.2.

In practice, the MWAL algorithm must rely on reinforcement learning algorithms that output only approximately optimal policies. Section 5.3 provides a unified analysis that takes this and related approximation errors into account. We prove that the performance of the MWAL algorithm degrades gracefully as the approximation error increases.

As in Section 4.4, our game-theoretic framework assumes that the true reward function can be described in terms of a set of known features. Section 5.4 explains that the choice of these features is critical, in the sense that changing them can strongly affect the policy learned by the apprentice. Our analysis also reveals that apprentice policies corresponding to lexicographic optimal strategies (see Chapter 3) are more robust to changes to the features.

A peculiar property of many algorithms for apprenticeship learning, including the MWAL algorithm, is that the policy learned by the apprentice has a complex and unnatural structure. Section 5.5 describes several techniques for converting these policies into equivalent simpler polices. One of the techniques leverages the lexicographic analysis from Chapter 3, while another is based on linear programming. In Section 5.6, we use linear programming in a slightly different way to derive a simple and direct apprenticeship learning algorithm that enjoys the same theoretical guarantees as the MWAL algorithm.

Section 5.7 describes experiments which compare the performance of several apprenticeship learning algorithms, both in terms of the quality of the policy learned by the apprentice, and the time required to learn it. These experiments reveal that our game-theoretic approach often produces better apprentice policies than algorithms that mimic the mentor, especially when the mentor is following a bad policy. Also, our lexicographic analysis from Chapter 3 provides an explanation for several of our experimental results that cannot be explained using existing analyses.

5.1 Modified Apprenticeship Learning Framework

In Chapter 4 we explained that the goal of apprenticeship learning is to find a mixed policy π_A such that $V(\pi_A) \geq V(\pi_E) - \epsilon$, even though the true reward function is unknown. In Section 4.4, we described algorithms that accomplish this goal by assuming that the true reward function is a linear combination of a set of known features, and then finding a (possibly mixed) apprentice policy that mimics the mentor by matching the feature expectations of the mentor's policy.

Nearly all aspects of the "feature-matching" framework of Section 4.4 will carry over to this chapter. We will continue to assume that the true reward function is a linear combination of the features, and that an apprenticeship learning algorithm is allowed to output a mixed policy. Also, all the algorithms we develop in this chapter will require access to subroutines that can perform standard reinforcement learning tasks, like computing an optimal policy for a given reward function.

However, some aspects of the framework from Section 4.4 will need to be modified to fit the game-theoretic approach of this chapter. First, we make a minor change to our assumptions about how the true reward function depends on the features. Second, and more significantly, the apprentice policy that we seek will not be one that mimics the mentor, but rather one that satisfies a certain game-theoretic objective. We

provide more details about these modifications in the rest of this section.

5.1.1 Feature Assumptions

Recall that in Section 4.4 we assumed that the true reward function R can be written

$$R(s) = \mathbf{w}^* \cdot \boldsymbol{\phi}(s)$$

where the *feature function* $\boldsymbol{\phi} : \mathcal{S} \to [0,1]^k$ is known, but the *weight vector* $\mathbf{w}^* \in \mathbb{R}^k$, where $\|\mathbf{w}^*\|_1 \leq 1$, is unknown. Our game-theoretic framework requires a change in our assumptions about $\boldsymbol{\phi}$ and $\mathbf{w}^*$. Specifically, we assume that $\boldsymbol{\phi}(s) \in [-1,+1]^k$ and $\mathbf{w}^* \in \Delta^k$, where Δ^k denotes the set of distributions on k elements.

In other words, under the original set of assumptions, the weight vector carries the sign of each feature, while under the new set of assumptions the features themselves carry the sign. These seemingly superficial differences actually have important consequences for the policies output by the algorithms, which we will discuss in Section 5.4.1. But in one sense, the assumptions are equivalent: The same class of reward functions can be expressed under either set of assumptions. Concretely, consider a reward function $R(s) = \mathbf{w} \cdot \boldsymbol{\phi}(s)$ such that $\boldsymbol{\phi}(s) \in [0,1]^k$ and $\|\mathbf{w}\|_1 \leq 1$. If we let

$$\boldsymbol{\phi}'(s) = (\phi_1(s), \ldots, \phi_k(s), -\phi_1(s), \ldots, -\phi_k(s), 0)$$

then $\boldsymbol{\phi}'(s) \in [-1,+1]$ and there clearly exists a weight vector $\mathbf{w}' \in \Delta^{2k+1}$ such that $R(s) = \mathbf{w}' \cdot \boldsymbol{\phi}'(s)$. So any reward function expressible under one set of assumptions is expressible under the other, at the expensive of roughly doubling the number of features.

5.1.2 Game-Theoretic Objective

Let $\mathbf{\Psi}$ be the set of all mixed policies. Now consider the optimization

$$\max_{\boldsymbol{\psi}\in\mathbf{\Psi}} \min_{\mathbf{w}\in\Delta^k} \left[\mathbf{w}\cdot\boldsymbol{\mu}(\boldsymbol{\psi}) - \mathbf{w}\cdot\boldsymbol{\mu}_E\right]. \tag{5.1}$$

Our goal will be to find (actually, to approximate) the mixed policy $\boldsymbol{\psi}_A$ that realizes this maximum. Since $V(\boldsymbol{\psi}) = \mathbf{w}^*\cdot\boldsymbol{\mu}(\boldsymbol{\psi})$ for all $\boldsymbol{\psi}$, we have that $\boldsymbol{\psi}_A$ is the mixed policy in $\mathbf{\Psi}$ that maximizes $V(\boldsymbol{\psi}) - V(\pi_E)$ with respect to the worst-case possibility for $\mathbf{w}^*$. Since $\mathbf{w}^*$ is unknown, maximizing for the worst-case is appropriate.

We want to put (5.1) in the form of a two-person zero-sum game. First, we shift and rescale (5.1) as follows:

$$\max_{\boldsymbol{\psi}\in\mathbf{\Psi}} \min_{\mathbf{w}\in\Delta^k} ((1-\gamma)(\mathbf{w}\cdot\boldsymbol{\mu}(\boldsymbol{\psi}) - \mathbf{w}\cdot\boldsymbol{\mu}_E) + 2)/4 \tag{5.2}$$

This formulation is equivalent in the sense that any mixed policy $\boldsymbol{\psi}_A$ which realizes the maximum in (5.2) also realizes the maximum in (5.1), and vice-versa. It follows from (4.6) that the value of (5.2) is in the interval $[0,1]$. Next we define a $k\times|\Pi_D|$ matrix

$$M(i,j) \triangleq ((1-\gamma)(\mu^j(i) - \mu_E(i)) + 2)/4 \tag{5.3}$$

where $\mu(i)$ is the ith component of $\boldsymbol{\mu}$ and we have let $\boldsymbol{\mu}^j = \boldsymbol{\mu}(\pi^j)$ be the vector of feature expectations for the jth deterministic policy π^j. Now (5.2) can be rewritten using (5.3) as

$$v^* \triangleq \max_{\boldsymbol{\psi}\in\mathbf{\Psi}} \min_{\mathbf{w}\in\Delta^k} \mathbf{w}\mathbf{M}\boldsymbol{\psi}. \tag{5.4}$$

Because $\mathbf{w}$ and $\boldsymbol{\psi}$ are both distributions, and because the entries of $\mathbf{M}$ are in the interval $[0,1]$, we have that (5.4) is in the form of a two-person zero-sum game, and thus v^* is the value of the game. In this game, the row player strategies and column player strategies correspond to reward functions and mixed policies, respectively. The

row player specifies a reward function by choosing $\mathbf{w}$, and the column player chooses a mixed policy $\boldsymbol{\psi}$. The goal of the row player is to cause the column player's policy to perform as poorly as possible relative to the mentor, and the column player's goal is just the opposite. Note that $\boldsymbol{\psi}_A$ is a maximin strategy in this game.

Finding the maximin strategy $\boldsymbol{\psi}_A$ will not be useful unless we can establish that $v^* \geq 0$, i.e. that $\boldsymbol{\psi}_A$ will do at least as well as the mentor's policy with respect to the worst-case possibility for $\mathbf{w}^*$. This fact is not immediately clear, since we are restricting ourselves to mixtures of deterministic policies, while we do not assume that the mentor's policy is deterministic.[1]

Theorem 5.1. *The quantity v^* in* (5.4) *is nonnegative.*

Proof. We know from the minimax theorem (Theorem 2.7) that we can swap the min and max operators in (5.4) without affecting the game value. In other words,

$$v^* = \max_{\boldsymbol{\psi} \in \boldsymbol{\Psi}} \min_{\mathbf{w} \in \Delta^k} \mathbf{w}\mathbf{M}\boldsymbol{\psi} = \min_{\mathbf{w} \in \Delta^k} \max_{\boldsymbol{\psi} \in \boldsymbol{\Psi}} \mathbf{w}\mathbf{M}\boldsymbol{\psi}. \tag{5.5}$$

Note that in the rightmost expression in Eq. (5.5), the maximization over $\boldsymbol{\Psi}$ is done after $\mathbf{w}$ has been fixed. By examining (5.2), we see that the maximum is achieved by the policy in $\boldsymbol{\psi}^* \in \boldsymbol{\Psi}$ which maximizes $\mathbf{w} \cdot \boldsymbol{\mu}(\boldsymbol{\psi})$. Because $V(\boldsymbol{\psi}) = \mathbf{w} \cdot \boldsymbol{\mu}(\boldsymbol{\psi})$ is the value of mixed policy $\boldsymbol{\psi}$ with respect to reward function $R = \mathbf{w} \cdot \boldsymbol{\phi}$, and because there always exists a deterministic optimal policy for any reward function (see discussion in Section 4.2.1, we have that $\mathbf{w} \cdot \boldsymbol{\mu}(\boldsymbol{\psi}^*)$ will be at least as large as $\mathbf{w} \cdot \boldsymbol{\mu}_E$. Hence $v^* \geq 0$. □

In fact, we may have $v^* > 0$. Let $\boldsymbol{\mu}_A = \boldsymbol{\mu}(\boldsymbol{\psi}_A)$, and suppose it happens that $\mu_A(i) > \mu_E(i)$ for all i. Then $\boldsymbol{\psi}_A$ *strictly dominates* π_E, i.e. $\boldsymbol{\psi}_A$ will have higher

[1] We could avoid this technical problem by allowing the apprenticeship learning algorithm to output a mixture of randomized policies. But that would require that the matrix defined in (5.3) have uncountably many columns. And that would, in turn, require a measure-theoretic foundation for our analysis which, while possible, would introduce significantly more complications than it would avoid.

value than π_E regardless of the actual value of $\mathbf{w}^*$, because we assumed that $\mathbf{w}^*(i)$ is nonnegative for all i. Essentially, by assuming that each component of the true weight vector is nonnegative, we are assuming that we have correctly specified the "sign" of each feature. This means that, other things being equal, a larger value for each feature implies a larger reward. So when $v^* > 0$, the mixed policy $\boldsymbol{\psi}_A$ to some extent ignores the mentor, and instead exploits prior knowledge about the true reward function encoded by the features. We discuss issues related to features in more detail in Section 5.4, and we present experimental results that explore this aspect of our approach in Section 5.7.

5.2 MWAL Algorithm

We can apply any of the game-solving techniques from Chapter 2 to compute the maximin strategy $\boldsymbol{\psi}_A$ that realizes (5.4). For example, we could find $\boldsymbol{\psi}_A$ by solving a linear program. However, the size of this linear program will scale with the size of the game matrix. In our case, the game matrix $\mathbf{M}$ is huge, since it has as many columns as the number of deterministic policies, $|\Pi_D| = |\mathcal{A}|^{|\mathcal{S}|}$.

As we described in Section 2.4.3, no-regret algorithms, such as the multiplicative weights (MW) algorithm due to Freund and Schapire [32], can be used to compute approximately optimal strategies in games with large game matrices. Applying MW to the game matrix $\mathbf{M}$ results in Algorithm 5.1, which we call MWAL (Multiplicative Weights for Apprenticeship Learning) [2] [127]. We immediately have the following guarantee.

Theorem 5.2. *Let $\overline{\boldsymbol{\psi}}_T$ be the mixed policy output by the MWAL algorithm in Algo-*

[2]There is one minor difference between the MWAL algorithm in Algorithm 5.1 and the description of MW in Section 2.4.3. In Algorithm 5.1, the weight vector $\mathbf{w}_t$ is renormalized in every iteration, while in Section 2.4.3, we used $\mathbf{p}_t$ to denote the renormalized version of $\mathbf{w}_t$. This difference is entirely cosmetic.

rithm 5.1. Then

$$\min_{\mathbf{w} \in \Delta^k} \mathbf{w}\mathbf{M}\overline{\boldsymbol{\psi}}_T \geq v^* - \Delta_{T,\mathbf{w}_1,\beta} \tag{5.6}$$

Proof. Recall that column player strategies in the game defined by $\mathbf{M}$ correspond to mixed policies. A direct application of Theorem 2.8 proves the statement of the theorem. □

Algorithm 5.1 MWAL Algorithm

1: **Given:** MDP without a reward function, feature function $\boldsymbol{\phi}$, mentor's feature expectations $\boldsymbol{\mu}_E$, parameters $\mathbf{w}_1, \beta$, and T.
2: **for** $t = 1, \ldots, T$ **do**
3: Normalize $\mathbf{w}_t$ to sum to 1.
4: Let $\boldsymbol{\psi}_t = \arg\max_{\boldsymbol{\psi} \in \boldsymbol{\Psi}} \mathbf{w}_t \mathbf{M} \boldsymbol{\psi}$.
5: Let $w_{t+1}(i) = w_t(i)\beta^{M(i,\boldsymbol{\psi}_t)}$ for each $i = 1, \ldots, k$.
6: **end for**
7: **Return:** $\overline{\boldsymbol{\psi}}_T = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\psi}_t$.

We can re-write the MWAL algorithm in a clearer way by observing that the maximum in Line 4 is attained by a single deterministic policy, and also observing that this maximum is unaffected by the mentor's policy. By "compiling out" the matrix $\mathbf{M}$, and also tuning the parameters β and $\mathbf{w}_1$ according to Lemma 2.5, we are left with Algorithm 5.2, which is an equivalent version of the MWAL algorithm. We also have the following more interpretable version of its guarantee.

Corollary 5.3. *Let $\overline{\boldsymbol{\psi}}_T$ be the mixed policy output by the MWAL algorithm in Algorithm 5.2. Then*

$$V(\overline{\boldsymbol{\psi}}_T) - V(\pi_E) \geq v^* - O\left(\sqrt{\frac{\log k}{T(1-\gamma)^2}}\right) \tag{5.7}$$

Proof. Consider

$$
\begin{aligned}
V(\overline{\boldsymbol{\psi}}_T) - V(\pi_E) &= \mathbf{w}^* \cdot \boldsymbol{\mu}(\overline{\boldsymbol{\psi}}_T) - \mathbf{w}^* \cdot \boldsymbol{\mu}_E && \text{((4.9) and (4.5))} \\
&\geq \min_{\mathbf{w} \in \Delta^k} \left[\mathbf{w} \cdot \boldsymbol{\mu}(\overline{\boldsymbol{\psi}}_T) - \mathbf{w} \cdot \boldsymbol{\mu}_E \right] && (\mathbf{w}^* \in \Delta^k) \\
&\geq v^* - \Delta_{T, \mathbf{w}_1, \beta}/(1-\gamma) && \text{(definition of } \mathbf{M} \text{ and Theorem 5.2)} \\
&\geq v^* - O\left(\sqrt{\frac{\log k}{T(1-\gamma)^2}} \right) && \text{(choice of } \beta \text{ and } \mathbf{w}_1 \text{ and Lemma 2.5)}
\end{aligned}
$$

□

Algorithm 5.2 MWAL Algorithm (equivalent version)

Given: MDP without a reward function, feature function $\boldsymbol{\phi}$, mentor's feature expectations $\boldsymbol{\mu}_E$, parameter T.
Let $\beta = \left(1 + \sqrt{\frac{2 \ln k}{T}}\right)^{-1}$.
Let $w_1(i) = 1$ for each $i = 1, \ldots, k$.
for $t = 1, \ldots, T$ **do**
    Normalize $\mathbf{w}_t$ to sum to 1.
    Let $\pi_t = \arg\max_{\pi \in \Pi_D} \mathbf{w}_t \cdot \boldsymbol{\mu}(\pi)$.
    Let $\boldsymbol{\mu}_t = \boldsymbol{\mu}(\pi_t)$.
    Let $x_t(i) = ((1-\gamma)(\mu_t(i) - \mu_E(i)) + 2)/4$ for each $i = 1, \ldots, k$.
    Let $w_{t+1}(i) = w_t(i)\beta^{x_t(i)}$ for each $i = 1, \ldots, k$.
end for
Return: The mixed policy $\overline{\boldsymbol{\psi}}_T$ given by the uniform distribution on $\pi_1, \ldots, \pi_T$.

5.2.1 Comparison to Feature-Matching Algorithms

The version of MWAL presented in Algorithm 5.2 is the easier version to compare directly to the Projection and Blackwell algorithms — the so-called "feature-matching" algorithms — from Section 4.4. By examining them, it is clear that all the algorithms have the same computational complexity per iteration, since they all perform the same tasks in each iteration: computing an optimal policy, and computing the feature expectations of that policy. Also, each algorithm requires the mentor's feature expectations $\boldsymbol{\mu}_E$.

However, by comparing the guarantees for the feature-matching and MWAL algorithms — see (4.10), (4.13) and (5.7) — we see that the MWAL algorithm requires many fewer iterations than the feature-matching algorithms to achieve the same error: $O(\log k)$ versus $O(k \log k)$ or $O(k)$ iterations, where k is the number of features. This represents a tremendous savings when the number of features is very large.

Also, the guarantee for the MWAL algorithm is a "one-sided" bound, while for the feature-matching algorithms it is a "two-sided" bound. In other words, while the performance of the mixed policies output by the MWAL and feature-matching algorithms are all guaranteed not to be much worse than the mentor's policy — which satisfies the goal of apprenticeship learning — the feature-matching algorithms' mixed policies are also guaranteed not to be much *better* than the mentor's policy. This is because the feature-matching algorithms essentially mimic the mentor. We present experimental results that demonstrate this advantage of the MWAL algorithm in Section 5.7.

Finally, like the Blackwell algorithm, the post-processing step for the MWAL algorithm is trivial, since the final mixed policy is just the uniform average of the policies computed over all T rounds, and unlike the Projection algorithm, no QP solver is required.

5.2.2 Absence of a Mentor

Unlike the feature-matching apprenticeship learning algorithms, the MWAL algorithm can be very naturally and easily extended to the case where we do not have observations from the mentor. Instead of finding a policy that maximizes the original apprenticeship learning objective in (5.1), we find a mixed policy $\boldsymbol{\psi}_A$ that maximizes

$$\max_{\boldsymbol{\psi} \in \boldsymbol{\Psi}} \min_{\mathbf{w} \in \Delta^k} \mathbf{w} \cdot \boldsymbol{\mu}(\boldsymbol{\psi}). \tag{5.8}$$

Here $\boldsymbol{\psi}_A$ is the best policy for the worst-case possibility for $\mathbf{w}^*$. In this sense, $\boldsymbol{\psi}_A$ is the "maximally conservative" policy. We describe experiments that compute such a policy in Section 5.7.

The MWAL algorithm can be trivially adapted to find the mixed policy $\boldsymbol{\psi}_A$ that realizes (5.8) just by setting $\boldsymbol{\mu}_E = \mathbf{0}$ (compare (5.8) to (5.1)). The following corollary follows immediately from the proof of Corollary 5.3.

Corollary 5.4. *Let $\overline{\boldsymbol{\psi}}_T$ be the mixed policy output by the MWAL algorithm in Algorithm 5.2, and suppose $\boldsymbol{\mu}_E = \mathbf{0}$. Then*

$$V(\overline{\boldsymbol{\psi}}_T) \geq v^* - O\left(\sqrt{\frac{\log k}{T(1-\gamma)^2}}\right) \tag{5.9}$$

5.3 Complete Analysis of MWAL

Our presentation of the MWAL algorithm has thus far assumed that the algorithm can exactly compute an optimal policy for any fixed reward function, as well as the feature expectations of that policy, and also has access to the feature expectations of the mentor's policy. In practice, all these quantities must be approximated. In this section, we complete the analysis of the MWAL algorithm by describing a version that uses only approximations, and show that the performance of this version degrades gracefully as the quality of these approximations decreases.

5.3.1 Approximation Version of MWAL

An "approximation" version of the MWAL algorithm — i.e. a version that uses only approximations of optimal policies, feature expectations, etc. — is given in Algorithm 5.3. In this section we review the differences between the approximation version of the MWAL algorithm and the "exact" version given in Algorithm 5.2.

In Step 2, the algorithm forms an estimate $\hat{\boldsymbol{\mu}}_E$ of the mentor's feature expec-

tations $\boldsymbol{\mu}_E$ by averaging the observed feature values from the mentor's trajectories. More precisely, let $(s_0^i, s_1^i, \ldots, s_H^i)$ denote the ith sample trajectory (for simplicity, we assume that all sample trajectories are truncated to the same length H). The estimate $\hat{\boldsymbol{\mu}}_E$ of $\boldsymbol{\mu}_E$ is

$$\hat{\boldsymbol{\mu}}_E = \frac{1}{m} \sum_{i=0}^{m} \sum_{t=0}^{H} \gamma^t \boldsymbol{\phi}(s_t^i). \tag{5.10}$$

Of course, both m and H will affect the quality of the estimate $\hat{\boldsymbol{\mu}}_E$. However, instead of working directly with H, it will be more convenient for us to let ϵ_H be a constant such that $H = (1/(1-\gamma)) \ln(1/(\epsilon_H(1-\gamma)))$.

In Step 7, the algorithm computes an ϵ-optimal policy. An ϵ-optimal policy $\hat{\pi}$ satisfies $V(\hat{\pi}) \geq V(\pi^*) - \epsilon$. In Section 4.2.1 we described several methods for computing such a policy in an MDP.

In Step 8, the algorithm computes an ϵ-good estimate of the feature expectations of a given policy. An ϵ-good estimate $\hat{\boldsymbol{\mu}}$ of $\boldsymbol{\mu}(\pi)$ satisfies $\|\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}(\pi)\|_\infty \leq \epsilon$. In Section 4.2.2 we described several methods for computing estimates of the value of a policy, and these methods can be trivially extended to compute an estimate of the feature expectations.

Let $\epsilon_R = \min_{\mathbf{w} \in \Delta^k} \max_s |R(s) - \mathbf{w} \cdot \boldsymbol{\phi}(s)|$ be the *representation error* of the features. We have thus far assumed that the representation error is zero, but our analysis in the next section will show that even this requirement can be relaxed.

5.3.2 Guarantee for Approximation Version

This section is devoted to proving the following guarantee about Algorithm 5.3.

Theorem 5.5. *Let $\overline{\boldsymbol{\psi}}$ be the mixed policy output by the MWAL algorithm in Algorithm 5.3. Then in order for*

$$V(\overline{\boldsymbol{\psi}}) \geq V(\pi_E) + v^* - \epsilon \tag{5.11}$$

Algorithm 5.3 MWAL Algorithm (approximation version)

Given: MDP without a reward function, feature function $\boldsymbol{\phi}$, a set of m independent trajectories from mentor's policy π_E, parameters T, ϵ_P, ϵ_F.
Form estimate $\hat{\boldsymbol{\mu}}_E$ of the mentor's feature expectations as in (5.10).
Let $\beta = \left(1 + \sqrt{\frac{2 \ln k}{T}}\right)^{-1}$.
Let $w_1(i) = 1$ for each $i = 1, \ldots, k$.
for $t = 1, \ldots, T$ **do**
    Normalize $\mathbf{w}_t$ to sum to 1.
    Let $\hat{\pi}_t$ be an ϵ_P-optimal policy for reward function $R_t(s) = \mathbf{w}_t \cdot \boldsymbol{\phi}(s)$.
    Let $\hat{\boldsymbol{\mu}}_t$ be an ϵ_F-good estimate of feature expectations $\boldsymbol{\mu}(\hat{\pi}_t)$.
    Let $x_t(i) = ((1-\gamma)(\hat{\mu}_t(i) - \hat{\mu}_E(i)) + 2)/4$ for each $i = 1, \ldots, k$.
    Let $w_{t+1}(i) = w_t(i)\beta^{x_t(i)}$ for each $i = 1, \ldots, k$.
end for
Return: The mixed policy $\overline{\boldsymbol{\psi}}_T$ given by the uniform distribution on $\hat{\pi}_1, \ldots, \hat{\pi}_T$.

to hold with probability at least $1 - \delta$*, it suffices that*

$$T \geq \frac{9 \ln k}{2(\epsilon'(1-\gamma))^2} \tag{5.12}$$

$$m \geq \frac{2}{(\epsilon'(1-\gamma))^2} \ln \frac{2k}{\delta} \tag{5.13}$$

$$\tag{5.14}$$

where

$$\epsilon' \leq \frac{\epsilon - (2\epsilon_F + \epsilon_P + 2\epsilon_H + 2\epsilon_R/(1-\gamma))}{3}. \tag{5.15}$$

The heart of the proof of Theorem 5.5 is the following corollary.

Corollary 5.6. *At the end of Algorithm 5.3*

$$\frac{1}{T}\sum_{t=1}^{T} [\mathbf{w}_t \cdot \hat{\boldsymbol{\mu}}_t - \mathbf{w}_t \cdot \hat{\boldsymbol{\mu}}_E] \leq \frac{1}{T} \min_{\mathbf{w} \in \Delta^k} \sum_{t=1}^{T} [\mathbf{w} \cdot \hat{\boldsymbol{\mu}}_t - \mathbf{w} \cdot \hat{\boldsymbol{\mu}}_E] + \Delta_{T,\mathbf{w}_1,\beta}/(1-\gamma)$$

Proof. For each possible policy $\pi^j \in \Pi^D$, fix an ϵ_F-good estimate $\hat{\boldsymbol{\mu}}^j$ of $\boldsymbol{\mu}(\pi^j)$, and assume without loss of generality that this is the estimate found by the algorithm. We are free to assume that $\hat{\boldsymbol{\mu}}^j \in [0, \frac{1}{1-\gamma}]^k$, because we can always trim $\hat{\boldsymbol{\mu}}^j$ so that it falls within the desired range without increasing the error in the estimate. Now

consider the matrix $\hat{\mathbf{M}}$, where

$$\hat{M}(i,j) = ((1-\gamma)(\hat{\mu}^j(i) - \hat{\mu}_E(i)) + 2)/4$$

Note that $\hat{M}(i,j) \in [0,1]$, and that the weights $\mathbf{w}_t$ in Algorithm 5.3 are updated according to the MW algorithm (see Section 2.4.3) when run on the matrix $\hat{\mathbf{M}}$. A direct application of Corollary 2.4 establishes the desired inequality. □

To complete the proof of Theorem 5.5, we will need to prove several auxiliary results. The next lemma bounds the number of trajectories needed to make $\hat{\boldsymbol{\mu}}_E$ close to $\boldsymbol{\mu}_E$.

Lemma 5.7. *For $\|\hat{\boldsymbol{\mu}}_E - \boldsymbol{\mu}_E\|_\infty \leq \epsilon + \epsilon_H$ to hold with probability at least $1-\delta$, it suffices that*

$$m \geq \frac{2}{(\epsilon(1-\gamma))^2} \ln\left(\frac{2k}{\delta}\right)$$

Proof. This is a standard proof using Hoeffding's inequality. However, care must be taken in one respect: $\hat{\boldsymbol{\mu}}_E$ is *not* an unbiased estimate of $\boldsymbol{\mu}_E$, because the trajectories are truncated at H. So define

$$\boldsymbol{\mu}_E^H \triangleq E\left[\sum_{t=0}^{H} \gamma^t \boldsymbol{\phi}(s_t) \;\middle|\; s_0 \sim \alpha(\cdot), a_t \sim \pi_E(s,\cdot), s_{t+1} \sim \theta(s_t, a_t, \cdot)\right]$$

Then we have,

$$\begin{aligned}
\forall i \in \{1,\ldots,k\} \quad & \Pr(|\hat{\mu}_E(i) - \mu_E^H(i)| \geq \epsilon) \leq 2\exp(-m(\epsilon(1-\gamma))^2/2) \\
\Rightarrow \quad & \Pr(\exists i \in \{1,\ldots,k\} \text{ s.t. } |\hat{\mu}_E(i) - \mu_E^H(i)| \geq \epsilon) \leq 2k\exp(-m(\epsilon(1-\gamma))^2/2) \\
\Rightarrow \quad & \Pr(\forall i \in \{1,\ldots,k\}, |\hat{\mu}_E(i) - \mu_E^H(i)| \leq \epsilon) \geq 1 - 2k\exp(-m(\epsilon(1-\gamma))^2/2) \\
\Rightarrow \quad & \Pr(\left\|\hat{\mu}_E - \mu_E^H\right\|_\infty \leq \epsilon) \geq 1 - 2k\exp(-m(\epsilon(1-\gamma))^2/2)
\end{aligned}$$

We used in order: Hoeffding's inequality and $\boldsymbol{\mu}_E^H \in [0, \frac{1}{1-\gamma}]^k$; the union bound; the

probability of disjoint events; the definition of L_∞ norm.

It is not hard to show that $\left\|\boldsymbol{\mu}_E^H - \boldsymbol{\mu}_E\right\|_\infty \leq \epsilon_H$ (see Kearns and Singh [56]). Hence if $m \geq \frac{2}{(\epsilon(1-\gamma))^2} \ln(\frac{2k}{\delta})$, then with probabilty at least $1-\delta$ we have

$$\|\hat{\boldsymbol{\mu}}_E - \boldsymbol{\mu}_E\|_\infty \leq \left\|\hat{\boldsymbol{\mu}}_E - \boldsymbol{\mu}_E^H\right\|_\infty + \left\|\boldsymbol{\mu}_E^H - \boldsymbol{\mu}_E\right\|_\infty \leq \epsilon + \epsilon_H$$

□

The next lemma bounds the impact of representation error: It says that if $R(s)$ and $\mathbf{w}^* \cdot \boldsymbol{\phi}(s)$ are not very different, then neither are $V(\boldsymbol{\psi})$ and $\mathbf{w}^* \cdot \boldsymbol{\mu}(\boldsymbol{\psi})$.

Lemma 5.8. *$|V(\boldsymbol{\psi}) - \mathbf{w}^* \cdot \boldsymbol{\mu}(\boldsymbol{\psi})| \leq \frac{\epsilon_R}{1-\gamma}$ for every mixed policy $\boldsymbol{\psi} \in \boldsymbol{\Psi}$.*

Proof.

$$\begin{aligned}
& |V(\boldsymbol{\psi}) - \mathbf{w}^* \cdot \boldsymbol{\mu}(\boldsymbol{\psi})| \\
= & \left| E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right] - E\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{w}^* \cdot \boldsymbol{\phi}(s_t)\right] \right| \\
= & \left| \lim_{H\to\infty} E\left[\sum_{t=0}^{H} \gamma^t R(s_t)\right] - \lim_{H\to\infty} E\left[\sum_{t=0}^{H} \gamma^t \mathbf{w}^* \cdot \boldsymbol{\phi}(s_t)\right] \right| \\
= & \left| \lim_{H\to\infty} E\left[\sum_{t=0}^{H} \gamma^t (R(s_t) - \mathbf{w}^* \cdot \boldsymbol{\phi}(s_t))\right] \right| \\
\leq & \lim_{H\to\infty} E\left[\sum_{t=0}^{H} \gamma^t |R(s_t) - \mathbf{w}^* \cdot \boldsymbol{\phi}(s_t)|\right] \\
\leq & \frac{\epsilon_R}{1-\gamma}
\end{aligned}$$

□

We are now ready to prove Theorem 5.5.

Proof of Theorem 5.5. Let $\overline{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{w}_t$. Then we have

$$
\begin{aligned}
v^* &= \max_{\boldsymbol{\psi}\in\boldsymbol{\Psi}} \min_{\mathbf{w}\in\Delta^k} [\mathbf{w}\cdot\boldsymbol{\mu}(\boldsymbol{\psi}) - \mathbf{w}\cdot\boldsymbol{\mu}_E] \\
&= \min_{\mathbf{w}\in\Delta^k} \max_{\boldsymbol{\psi}\in\boldsymbol{\Psi}} [\mathbf{w}\cdot\boldsymbol{\mu}(\boldsymbol{\psi}) - \mathbf{w}\cdot\boldsymbol{\mu}_E] && (5.16) \\
&\leq \min_{\mathbf{w}\in\Delta^k} \max_{\boldsymbol{\psi}\in\boldsymbol{\Psi}} [\mathbf{w}\cdot\boldsymbol{\mu}(\boldsymbol{\psi}) - \mathbf{w}\cdot\hat{\boldsymbol{\mu}}_E] + \epsilon' + \epsilon_H && (5.17) \\
&\leq \max_{\boldsymbol{\psi}\in\boldsymbol{\Psi}} [\overline{\mathbf{w}}\cdot\boldsymbol{\mu}(\boldsymbol{\psi}) - \overline{\mathbf{w}}\cdot\hat{\boldsymbol{\mu}}_E] + \epsilon' + \epsilon_H \\
&= \max_{\boldsymbol{\psi}\in\boldsymbol{\Psi}} \frac{1}{T}\sum_{t=1}^{T} [\mathbf{w}_t\cdot\boldsymbol{\mu}(\boldsymbol{\psi}) - \mathbf{w}_t\cdot\hat{\boldsymbol{\mu}}_E] + \epsilon' + \epsilon_H && (5.18) \\
&\leq \frac{1}{T}\sum_{t=1}^{T} \max_{\boldsymbol{\psi}\in\boldsymbol{\Psi}} [\mathbf{w}_t\cdot\boldsymbol{\mu}(\boldsymbol{\psi}) - \mathbf{w}_t\cdot\hat{\boldsymbol{\mu}}_E] + \epsilon' + \epsilon_H \\
&\leq \frac{1}{T}\sum_{t=1}^{T} [\mathbf{w}_t\cdot\boldsymbol{\mu}(\hat{\pi}_t) - \mathbf{w}_t\cdot\hat{\boldsymbol{\mu}}_E] + \epsilon_P + \epsilon' + \epsilon_H && (5.19) \\
&\leq \frac{1}{T}\sum_{t=1}^{T} [\mathbf{w}_t\cdot\hat{\boldsymbol{\mu}}_t - \mathbf{w}_t\cdot\hat{\boldsymbol{\mu}}_E] + \epsilon_F + \epsilon_P + \epsilon' + \epsilon_H && (5.20) \\
&\leq \frac{1}{T}\min_{\mathbf{w}\in\Delta^k} \sum_{t=1}^{T} [\mathbf{w}\cdot\hat{\boldsymbol{\mu}}_t - \mathbf{w}\cdot\hat{\boldsymbol{\mu}}_E] + \frac{\Delta_{T,\mathbf{w}_1,\beta}}{(1-\gamma)} + \epsilon_F + \epsilon_P + \epsilon' + \epsilon_H && (5.21) \\
&\leq \frac{1}{T}\min_{\mathbf{w}\in\Delta^k} \sum_{t=1}^{T} [\mathbf{w}\cdot\boldsymbol{\mu}(\hat{\pi}_t) - \mathbf{w}\cdot\hat{\boldsymbol{\mu}}_E] + \frac{\Delta_{T,\mathbf{w}_1,\beta}}{(1-\gamma)} + 2\epsilon_F + \epsilon_P + \epsilon' + \epsilon_H && (5.22) \\
&= \min_{\mathbf{w}\in\Delta^k} \left[\mathbf{w}\cdot\boldsymbol{\mu}(\overline{\boldsymbol{\psi}}) - \mathbf{w}\cdot\hat{\boldsymbol{\mu}}_E\right] + \frac{\Delta_{T,\mathbf{w}_1,\beta}}{(1-\gamma)} + 2\epsilon_F + \epsilon_P + \epsilon' + \epsilon_H && (5.23) \\
&\leq \min_{\mathbf{w}\in\Delta^k} \left[\mathbf{w}\cdot\boldsymbol{\mu}(\overline{\boldsymbol{\psi}}) - \mathbf{w}\cdot\boldsymbol{\mu}_E\right] + \frac{\Delta_{T,\mathbf{w}_1,\beta}}{(1-\gamma)} + 2\epsilon_F + \epsilon_P + 2\epsilon' + 2\epsilon_H && (5.24) \\
&\leq \mathbf{w}^*\cdot\boldsymbol{\mu}(\overline{\boldsymbol{\psi}}) - \mathbf{w}^*\cdot\boldsymbol{\mu}_E + \frac{\Delta_{T,\mathbf{w}_1,\beta}}{(1-\gamma)} + 2\epsilon_F + \epsilon_P + 2\epsilon' + 2\epsilon_H && (5.25) \\
&\leq V(\overline{\boldsymbol{\psi}}) - V(\pi_E) + \frac{\Delta_{T,\mathbf{w}_1,\beta}}{(1-\gamma)} + 2\epsilon_F + \epsilon_P + 2\epsilon' + 2\epsilon_H + \frac{2\epsilon_R}{(1-\gamma)} && (5.26)
\end{aligned}
$$

In (5.16), we used von Neumann's minmax theorem. In (5.17), Lemma 5.7. In (5.18), the definition of $\overline{\mathbf{w}}$. In (5.19), the fact that $\hat{\pi}_t$ is ϵ_P-optimal with respect to $R(s) = \mathbf{w}^t\cdot\phi(s)$. In (5.20), the fact that $\hat{\boldsymbol{\mu}}_t$ is an ϵ_F-good estimate of $\boldsymbol{\mu}(\hat{\pi}_t)$. In (5.21), Corollary 5.6. In (5.22), again the fact that $\hat{\boldsymbol{\mu}}_t$ is an ϵ_F-good estimate of $\boldsymbol{\mu}(\hat{\pi}_t)$. In (5.23), the definition of $\overline{\boldsymbol{\psi}}$. In (5.24), Lemma 5.7. In (5.25), we let

$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\Delta^k} \max_s |R(s) - (\mathbf{w}\cdot\boldsymbol{\phi}(s))|$. In (5.26), Lemma 5.8.

Plugging in the choice for T, $\mathbf{w}_1$ and β into $\Delta_{T,\mathbf{w}_1,\beta}$ and rearranging implies the theorem. □

5.4 Issues Related to Features

In our game-theoretic formulation of the apprenticeship learning problem, we have assumed that the true reward function can be described in terms of a set of known features. In this section, we explain that the choice of these features is critical, in the sense that changing them can strongly affect the policy learned by the apprentice.

5.4.1 Prior Knowledge

Though it may not be immediately obvious, the assumption that the true weight vector $\mathbf{w}^*$ contains only positive weights (i.e. that $\mathbf{w}^* \in \Delta^k$) has non-trivial consequences. Let

$$\boldsymbol{\phi}(s) = (\phi_1(s), \ldots, \phi_k(s))$$

be the features at state s. If $\phi_i(s) \geq \phi_i(s')$ for all features i, then $R(s) \geq R(s')$, regardless of the value of the true weight vector $\mathbf{w}^*$. Put differently, the positive weight assumption implies that all features are positively correlated with reward. If a feature is not positively correlated with reward — for example, in the car driving simulator, this might be a feature whose value increases with the number of crashes — then it should be negated before it is used by the MWAL algorithm.

There will be occasions, of course, when the correct "sign" of the relationship between a feature and the reward function is not known, and we wish to abstain from having to specify it. Our framework can easily accommodate this requirement. Let $\phi_{i_1}, \ldots, \phi_{i_\ell}$ be the indices of all features whose correct sign is unknown. Then we can

define a new feature function

$$\boldsymbol{\phi}'(s) = (\phi_1(s), \ldots, \phi_k, -\phi_{i_1}(s), \ldots, -\phi_{i_\ell}(s))$$

The effect of adding these extra features is equivalent to allowing the weights assigned to features $\phi_{i_1}, \ldots, \phi_{i_\ell}$ to be positive *or* negative. Also note that this modification at most doubles the number of features.

Another effect of adding negated versions of features is that it forces the apprentice policy to *mimic* the behavior of the mentor with respect to those features, much like the Projection algorithm of Abbeel and Ng [1]. The next theorem proves this.

Theorem 5.9. *Let $\boldsymbol{\psi}_A$ be a mixed policy that achieves v^* in* (5.4)*. Let $\boldsymbol{\mu}_A = \boldsymbol{\mu}(\boldsymbol{\psi}_A)$ be the feature expectations of $\boldsymbol{\psi}_A$. If there exist features ϕ_i and ϕ_j such that $\phi_i(s) = -\phi_j(s)$ for all states s, then $\mu_A(i) = \mu_E(i)$ and $\mu_A(j) = \mu_E(j)$.*

Proof. By the linearity of expectation we have $\mu_A(i) = -\mu_A(j)$ and $\mu_E(i) = -\mu_E(j)$. Thus, if the theorem does not hold, then either

$$\mu_A(i) > \mu_E(i) \text{ and } \mu_A(j) < \mu_E(j) \tag{5.27}$$

or

$$\mu_A(i) < \mu_E(i) \text{ and } \mu_A(j) > \mu_E(j) \tag{5.28}$$

Suppose for contradiction that (5.27) is the case, and let $\mathbf{w}_j$ be a weight vector that places all weight on feature j. Then $v^* < 0$, because $\mathbf{w}_j \cdot \boldsymbol{\mu}_A - \mathbf{w}_j \cdot \boldsymbol{\mu}_E < 0$. But this contradicts Theorem 5.1, which proved that $v^* \geq 0$. By a symmetric argument, (5.28) also contradicts Theorem 5.1. □

Theorem 5.9 tells us that, if we wish to find an apprentice policy that performs better than the mentor, then we must specify some features which are positively correlated with reward.

The number of features k can also be viewed as a measure of how much the apprentice knows about the true reward function. If $k = 1$, the (only) feature expectation of a policy is equal to its value, and thus the problem of finding a good apprentice policy reduces to finding an optimal policy in a standard MDP.

At the other extreme, consider a set of $2|\mathcal{S}|$ features defined by

$$\boldsymbol{\phi}(s) = \left(I(s = s_1), \ldots, I(s = s_{|\mathcal{S}|}), -I(s = s_1), \ldots, -I(s = s_{|\mathcal{S}|})\right), \tag{5.29}$$

where I is the indicator function. Note that for *any* reward function R, there exists a $\mathbf{w}^* \in \Delta^{2|\mathcal{S}|}$ such that $R(s) \propto \mathbf{w}^* \cdot \boldsymbol{\phi}(s)$. In this situation, the apprentice knows essentially nothing about the true reward function, and will be compelled to mimic the mentor.

5.4.2 Rescaling Units

In the apprenticeship learning framework, the features $\boldsymbol{\phi}$ encode what the reward function depends on, while the (unknown) weight vector $\mathbf{w}^*$ encodes how the features relate to each other, i.e., how the one should trade-off among them. The purpose of dividing the reward function in this manner is to cleanly separate what is easy to specify about the reward function from what is not. However, is this division actually as successful as it appears to be? Specifically, what impact does the choice of "units" for the features have on the maximin strategy in the apprenticeship learning game? For example, in the car driving example, suppose the feature for speed is measured in k.p.h. instead of m.p.h. While seemingly inconsequential, this change will shift all the values for the speed feature into a higher range. Will the mixed policy $\boldsymbol{\psi}_A$, which is a maximin strategy, drive faster as a result?

To make our discussion precise, suppose each feature ϕ_i is passed through a linear transformation L_i, which represents a change in "units". Consider a game matrix $\mathbf{M}_L$,

which is defined in the same way as the apprenticeship learning game matrix $\mathbf{M}$ in (5.3), except with respect to the new features $L_i(\phi_i)$. By the linearity of expectation, we have

$$M_L(i,j) = L_i(M(i,j)) \tag{5.30}$$

So $\mathbf{M}_L$ can be derived directly from $\mathbf{M}$ by applying the function L_i to each element of the ith row of $\mathbf{M}$. For the sake of generality, in the rest of this section we will suspend the apprenticeship learning motivation, and simply ask the following question about zero-sum games: If a different linear transformation is applied to each row of a game matrix $\mathbf{M}$, as in (5.30), under what conditions are the maximin strategies preserved?

Arbitrary linear transformations can, in fact, substantially affect the maximin strategies. Consider the following 3×4 game matrix, taken from the proof of Theorem 3.7:

$$\mathbf{M} = \begin{bmatrix} a+\epsilon & a-\epsilon & a+2\epsilon & a-2\epsilon \\ a-\epsilon & a+\epsilon & a-2\epsilon & a+2\epsilon \\ 2a & 2a & a & a \end{bmatrix}$$

where $a \in (0, \frac{1}{2}]$ and $\epsilon \in (0, \frac{a}{4})$. It is straightforward to verify that a is the value of the game defined by the matrix $\mathbf{M}$, and therefore the column strategy $\mathbf{q}^* = \left[0, 0, \frac{1}{2}, \frac{1}{2}\right]^T$ is a maximin strategy, since

$$\mathbf{M}\mathbf{q}^* = \begin{bmatrix} a \\ a \\ a \end{bmatrix}$$

Now consider the linear functions $L_1(x) = L_2(x) = cx$ and $L_3(x) = x$, where $c > 1$, and define the matrix $\mathbf{M}_L$ as in (5.30). By linearity, we have

$$\mathbf{M}_L\mathbf{q}^* = \begin{bmatrix} ca \\ ca \\ a \end{bmatrix}$$

However, the value of the game defined by the matrix $\mathbf{M}_L$ is $\min\{ca, 2a\}$, since by letting $\mathbf{q}_L^* = \left[\frac{1}{2}, \frac{1}{2}, 0, 0\right]^T$ we have

$$\mathbf{M}_L \mathbf{q}_L^* = \begin{bmatrix} ca \\ ca \\ 2a \end{bmatrix}$$

Thus $\mathbf{q}^*$ is not a maximin strategy for $\mathbf{M}_L$, while $\mathbf{q}_L^*$ is. Note that $\mathbf{q}_L^*$ is also a *lexicographic* maximin strategy for the original game matrix $\mathbf{M}$. So in this case, the lexicographic maximin strategy appears to be a better choice for the column player, since it is insensitive to the perturbations $L_1, \ldots, L_3$ described above. In this rest of this section, we will prove that this fact about lexicographic maximin strategies holds more generally: Such strategies are provably more robust to certain changes in the game matrix. This further validates the goal of computing a lexicographic maximin strategy.

For any game matrix $\mathbf{M}$, recall from Chapter 3 the definition

$$\mathbf{v}^* \triangleq \text{lexmax}_{\mathbf{q} \in \mathbb{Q}} \boldsymbol{\theta}(\mathbf{M}\mathbf{q}) \tag{5.31}$$

As we did in Chapter 3, we assume without loss of generality that the rows of $\mathbf{M}$ are ordered so that

$$\mathbf{v}^* = \mathbf{M}\mathbf{q}^{\text{lex}\,*} \tag{5.32}$$

where $\mathbf{q}^{\text{lex}\,*}$ is a lexicographic maximin strategy that achieves the objective in (5.31).

By definition, $v^*(1) \leq \cdots \leq v^*(n)$. As in Section 3.7.1, let $\tilde{\imath}$ be the largest row index such that $v^*(\tilde{\imath}) = v^*(1)$. This implies that $v^*(1) = \cdots = v^*(\tilde{\imath}) < v^*(\tilde{\imath} + 1) \leq \cdots \leq v^*(n)$. The next theorem shows, essentially, that any linear functions $L_1, \ldots, L_n$ which preserve this relationship among the components of $\mathbf{v}^*$ will not affect the optimality of a lexicographic maximin strategy.

Theorem 5.10. *Let* $\mathbf{M}$ *be an* $n \times m$ *game matrix, and let* $L_1, \ldots, L_n$ *be linear functions mapping* $\mathbb{R}$ *to* $\mathbb{R}$*. Define the matrix* $\mathbf{M}_L$ *as in* (5.30)*. Let* $\mathbf{q}^{\text{lex}\,*}$ *be a lexicographic maximin strategy for the matrix* $\mathbf{M}$*, and let* $\mathbf{q}_L^*$ *be a maximin strategy for the matrix* $\mathbf{M}_L$*.*

If $L_1 = \cdots = L_{\tilde{\imath}}$ *and* $L_j(v^*(j)) > L_{\tilde{\imath}}(v^*(\tilde{\imath}))$ *for all* $j > \tilde{\imath}$*, then*

$$\min_{\mathbf{p} \in \mathbb{P}} \mathbf{p}\mathbf{M}_L\mathbf{q}^{\text{lex}\,*} \geq \min_{\mathbf{p} \in \mathbb{P}} \mathbf{p}\mathbf{M}_L\mathbf{q}_L^*$$

Proof. We have

$$L_{\tilde{\imath}}(v^*(\tilde{\imath})) = \min_i L_i(v^*(i)) = \min_i L_i(M(i, \mathbf{q}^{\text{lex}\,*})) = \min_i \mathbf{M}_L(i, \mathbf{q}^{\text{lex}\,*}) \tag{5.33}$$

where the first equality uses the assumptions about $L_1, \ldots, L_n$, and the second and third equality use (5.32) and (5.30), respectively. Now suppose for contradiction that

$$\min_{\mathbf{p} \in \mathbb{P}} \mathbf{p}\mathbf{M}_L\mathbf{q}^{\text{lex}\,*} < \min_{\mathbf{p} \in \mathbb{P}} \mathbf{p}\mathbf{M}_L\mathbf{q}_L^*$$

which implies that

$$\min_i \mathbf{M}_L(i, \mathbf{q}^{\text{lex}\,*}) < \min_i \mathbf{M}_L(i, \mathbf{q}_L^*) = \min_i L_i(M(i, \mathbf{q}_L^*)) \tag{5.34}$$

Combining (5.33) and (5.34) yields $L_{\tilde{\imath}}(v^*(\tilde{\imath})) < \min_i L_i(M(i, \mathbf{q}_L^*))$. In particular, we have $L_{\tilde{\imath}}(v^*(\tilde{\imath})) < L_i(M(i, \mathbf{q}_L^*))$ for all $i \in \{1, \ldots, \tilde{\imath}\}$. And by our assumption about $L_1, \ldots, L_{\tilde{\imath}}$ and the definition of $\tilde{\imath}$, this implies that $v^*(i) < M(i, \mathbf{q}_L^*)$ for all $i \in \{1, \ldots, \tilde{\imath}\}$.

We will now show that there exists a $\lambda \in [0, 1]$ such that $\mathbf{q}_\lambda = \lambda\mathbf{q}_L^* + (1 - \lambda)\mathbf{q}^{\text{lex}\,*}$ and $\min_{\mathbf{p} \in \mathbb{P}} \mathbf{p}\mathbf{M}\mathbf{q}_\lambda > v^*(\tilde{\imath})$, which is a contradiction, since in that case $\mathbf{q}_\lambda$ would guarantee a higher payoff to the column player than any maximin strategy.

Recall that $v^*(i) = M(i, \mathbf{q}^{\text{lex}\,*})$ for all i. We proved above that $v^*(i) < M(i, \mathbf{q}_L^*)$ for all $\{1, \ldots, \tilde{\imath}\}$. Therefore, if $\lambda > 0$, we have $v^*(i) < \lambda M(i, \mathbf{q}_L^*) + (1-\lambda) M(i, \mathbf{q}^{\text{lex}\,*}) = M(j, \mathbf{q}_\lambda)$ for all $i \in \{1, \ldots, \tilde{\imath}\}$. Moreover, as $\lambda \to 0$, the value of $M(i, \mathbf{q}_\lambda)$ becomes arbitrarily close to $v^*(i)$ for all i. And since $v^*(i) > v^*(\tilde{\imath})$ for all $i > \tilde{\imath}$ (by definition of $\tilde{\imath}$), we can choose a value for λ such that $\min_{\mathbf{p} \in \mathbb{P}} \mathbf{pMq}_\lambda > v^*(\tilde{\imath})$. □

5.5 Outputting Stationary Policies

In Sections 4.4.2 and 5.2 we explained that both the Blackwell and MWAL algorithms output a mixed policy $\overline{\boldsymbol{\psi}}_T$, which is defined in terms of the set of the stationary policies $\{\pi_1, \ldots, \pi_T\}$ generated during the course of the algorithm. In particular, $\overline{\boldsymbol{\psi}}_T$ assigns weight $\frac{1}{T}$ to the policy π_t output by in round t of the algorithm. In Section 4.4.1 we explained that the Projection algorithm due to Abbeel and Ng [1] also outputs a policy of this type, although the mixture weights are not necessarily uniform.

Mixed policies are not commonly output by standard reinforcement learning algorithms, and in some sense are quite unnatural. Consider how an agent would follow a mixed policy in practice: At time 0, the agent must perform a "lottery", randomly choosing a single stationary policy from the set $\{\pi_1, \ldots, \pi_T\}$. The performance guarantees for mixed policies that we have presented thus far only hold in expectation over the outcome of this lottery. And if the policies in the set are very different from each other, the behavior of the mixed policy will be difficult to understand, or even to describe. On the other hand, stationary policies have the simplest form one can expect: In each state, they prescribe a single action or distribution over actions.

In Section 5.5.1, we prove that, under certain technical conditions, one can use the *last* stationary policy π_T generated by the MWAL algorithm, instead of all of them, thereby circumventing the drawbacks of a mixed policies entirely. The proof follows directly from our results on the MW algorithm and lexicographic optimality from

Chapter 3. In Section 5.5.2, we show how the MWAL algorithm can be modified so that it outputs a single stationary policy in all circumstances. This stationary policy is essentially a *per-state* average of all the policies generated by the MWAL algorithm. The main technical difficulty is determining the contribution of each individual policy to the average at each state.

5.5.1 Strict Case

In Section 3.6, we defined the following quantity for any game matrix $\mathbf{M}$

$$\delta_{\mathbf{M}} \triangleq \min_{i \neq j} |v^*(i) - v^*(j)| \tag{5.35}$$

and proved that the MW algorithm converges to a lexicographic maximin strategy whenever $\delta_{\mathbf{M}} > 0$. Moreover, our proof applied to the *last* strategy output by the MW algorithm. Since every lexicographic maximin strategy is also a maximin strategy, and since the MWAL algorithm is just the MW algorithm applied to a particular game matrix, our results on lexicographic optimality imply the following corollary.

Corollary 5.11. *Suppose that Algorithm 5.1 is run for T rounds, and define $\delta_{\mathbf{M}}$ as in* (5.35)*. If $\delta_{\mathbf{M}} > 0$ then for all sufficiently large T*

$$V(\pi_T) - V(\pi_E) \geq v^* \tag{5.36}$$

Proof. This follows immediately from Theorem 3.9 and the definition of Algorithm 5.1. □

5.5.2 Dual Methods

As we explained in Section 5.2, the MWAL algorithm requires a subroutine that can compute an optimal policy in an MDP. A classic iterative technique such as value

iteration or policy iteration, which were described in Section 4.2.1, will typically be used in this role. In this section, we show that if one uses a certain linear program to find an MDP's optimal policy (one that is the dual of the linear program described in Section 4.2.1), then the MWAL algorithm can be modified so that it outputs a single stationary policy instead of a mixed policy. This technique preserves the performance guarantees of the MWAL algorithm, and can be applied even when the strictness condition used in the previous section does not hold. Moreover, the technique can be straightforwardly applied to *any* mixed policy — such as one output by the Projection algorithm or Blackwell algorithm — to convert it to a single stationary policy that has the same value. Importantly, no information about the true reward function is required. The material in the remainder of this section and the next section was first described by Syed and Schapire [128].

Dual Policy Representation

Recall that a policy π specifies the probability of taking each action in each state. In this section, we will introduce an equivalent representation for policies called the *occupancy measure*. This alternate representation will guide our development of a modified version of the MWAL algorithm that always outputs a stationary policy.

A policy π has occupancy measure $\mathbf{x}^\pi$ if[3]

$$x^\pi(s,a) = E\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{(s_t=s \wedge a_t=a)} \;\middle|\; s_0 \sim \alpha(\cdot), a_t \sim \pi(s_t,\cdot), s_{t+1} \sim \theta(s_t,a_t,\cdot)\right] \quad (5.37)$$

for all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$. In other words, $x^\pi(s,a)$ is the expected discounted number of visits to state-action pair (s,a) when following policy π.

[3]Here $\mathbf{x}^\pi$ is a vector whose (s,a)th component is denoted $x^\pi(s,a)$.

Now consider the following linear program:

$$\max_{\mathbf{x}} \sum_{s,a} R(s)x(s,a) \tag{5.38}$$

such that

$$\sum_{a} x(s,a) = \alpha(s) + \gamma \sum_{s',a} x(s',a)\theta(s',a,s) \qquad \forall s \in \mathcal{S} \tag{5.39}$$

$$x(s,a) \geq 0 \qquad \forall s \in \mathcal{S},\ \forall a \in \mathcal{A} \tag{5.40}$$

We will prove shortly that if $\mathbf{x}^*$ is a solution to (5.38) - (5.40) then

$$\pi^*(s,a) = \frac{x^*(s,a)}{\sum_a x^*(s,a)}$$

is an optimal policy, and $\mathbf{x}^*$ is the occupancy measure of π^*. This result is well-known, and can be found in most standard references on MDPs [100]. Often (5.39) - (5.40) are called the *Bellman flow constraints*.

The linear program in (5.38) - (5.40) is actually the dual of the linear program described in Section 4.2.1. Accordingly, solving (5.38) - (5.40) is often called the *Dual LP* method of solving MDPs .

Having found an optimal policy by the Dual LP method, computing its value and feature expectations is straightforward.

Lemma 5.12. *If policy π has occupancy measure $\mathbf{x}^\pi$, then $V(\pi) = \sum_{s,a} R(s)x^\pi(s,a)$ and $\boldsymbol{\mu}(\pi) = \sum_{s,a} \boldsymbol{\phi}(s)x^\pi(s,a)$.*

Proof. This is immediate from the definitions of the occupancy measure, feature expectations, and the value of a policy, plus the linearity of expectation. □

In fact, *all* $\mathbf{x}$ that satisfy the Bellman flow constraints (5.39) - (5.40) are the occupancy measure of *some* stationary policy, as the next theorem shows. Together with

Lemma 5.12, this theorem also proves that the solution to (5.38) - (5.40) corresponds to an optimal policy.

Theorem 5.13. *A vector* $\mathbf{x}$ *is the occupancy measure of a stationary policy* π *if and only if for all states* $s \in \mathcal{S}$ *and actions* $a \in \mathcal{A}$

$$\pi(s,a) = \frac{x(s,a)}{\sum_a x(s,a)}$$

and $\mathbf{x}$ *satisfies the Bellman flow constraints* (5.39) - (5.40)*.*

An equivalent result as Theorem 5.13 is given by Feinberg and Schwartz [25], but our proof below is much simpler and more direct. Interestingly, Lemma 5.12 and Theorem 5.13 prove the correctness of the Dual LP method of finding an optimal policy in an MDP, yet neither result requires the application of LP duality.

Before proceeding with the proof of Theorem 5.13, we introduce another linear system. For any stationary policy π, the *π-specific Bellman flow constraints* are given by the following linear system in which the variable $x(s,a)$ is unknown for all $(s,a) \in \mathcal{S} \times \mathcal{A}$:

$$x(s,a) = \pi(s,a)\alpha(s) + \pi(s,a)\gamma \sum_{s',a'} x(s',a')\theta(s',a',s) \tag{5.41}$$

$$x(s,a) \geq 0 \tag{5.42}$$

The next lemma shows that π-specific Bellman flow constraints have a solution.

Lemma 5.14. *For any stationary policy* π*, the occupancy measure* $\mathbf{x}^\pi$ *of* π *satisfies the* π*-specific Bellman flow constraints* (5.41) - (5.42)*.*

Proof. Clearly, $x^\pi(s,a)$ is non-negative for all s, a, and so (5.42) is satisfied. As for (5.41), we simply plug in the definition of $x^\pi(s,a)$ from (5.37). In the following derivation, all the expectations and probabilities are implicitly with respect to α, θ,

and π.

$$
\begin{aligned}
x^\pi(s,a) &= E\left[\sum_{t=0}^{\infty}\gamma^t \mathbf{1}_{(s_t=s\wedge a_t=a)}\right] \\
&= \sum_{t=0}^{\infty}\gamma^t \Pr(s_t=s, a_t=a) \\
&= \pi(s,a)\alpha(s) + \sum_{t=0}^{\infty}\gamma^{t+1}\Pr(s_{t+1}=s, a_{t+1}=a) \\
&= \pi(s,a)\alpha(s) + \sum_{t=0}^{\infty}\gamma^{t+1}\sum_{s',a'}\Pr(s_t=s', a_t=a', s_{t+1}=s, a_{t+1}=a) \\
&= \pi(s,a)\alpha(s) + \sum_{t=0}^{\infty}\gamma^{t+1}\sum_{s',a'}\Pr(s_t=s', a_t=a')\cdot\theta(s',a',s)\pi(s,a) \\
&= \pi(s,a)\alpha(s) + \pi(s,a)\gamma\sum_{s',a'}E\left[\sum_{t=0}^{\infty}\gamma^t\mathbf{1}_{(s_t=s'\wedge a_t=a')}\right]\theta(s',a',s) \\
&= \pi(s,a)\alpha(s) + \pi(s,a)\gamma\sum_{s',a'}x^\pi_{s'a'}\theta(s',a',s)
\end{aligned}
$$

□

Now we show that the solution to the π-specific Bellman flow constraints given by Lemma 5.14 is unique.

Lemma 5.15. *For any stationary policy π, the π-specific Bellman flow constraints* (5.41) - (5.42) *have at most one solution.*

Proof. Define the matrix $\mathbf{A}$ and vector $\mathbf{b}$ as follows.[4] Let

$$
A((s,a),(s',a')) \triangleq \begin{cases} 1-\gamma\theta(s',a',s)\pi(s,a) & \text{if } (s,a)=(s',a') \\ -\gamma\theta(s',a',s)\pi(s,a) & \text{otherwise.} \end{cases}
$$

[4]Here $\mathbf{A}$ and $\mathbf{b}$ are indexed by state-action pairs, following the same indexing convention as $\mathbf{x}$.

and the vector $b(s,a) \triangleq \pi(s,a)\alpha(s)$. We can re-write (5.41) - (5.42) equivalently as

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{5.43}$$

$$\mathbf{x} \geq 0 \tag{5.44}$$

where the inequality should be interpreted component-wise. We claim that the matrix $\mathbf{A}$ is column-wise strictly diagonally dominant. Note that $\sum_{s'} \theta(s,a,s') = 1$, $\sum_a \pi(s,a) = 1$ and $\gamma < 1$, so for all $s' \in \mathcal{S}$ and $a' \in \mathcal{A}$

$$\sum_{s,a} \gamma\theta(s',a',s)\pi(s,a) = \gamma < 1$$

Rearranging this inequality yields

$$1 - \gamma\theta(s',a',s')\pi(s',a') > \sum_{(s,a)\neq(s',a')} \gamma\theta(s',a',s)\pi(s,a)$$

And substituting the definition of $\mathbf{A}$ we have

$$|A((s',a'),(s',a'))| > \sum_{(s,a)\neq(s',a')} |A((s,a),(s',a'))|$$

The last inequality is the definition of column-wise strict diagonal dominance. This implies that $\mathbf{A}$ is non-singular [49], so (5.43) - (5.44) has at most one solution. □

We are now ready to prove Theorem 5.13.

Proof of Theorem 5.13. For one direction of the theorem, we assume that $\mathbf{x}$ satisfies the Bellman flow constraints (5.39) - (5.40), and that $\pi(s,a) = \dfrac{x(s,a)}{\sum_{a'} x(s,a')}$. Therefore,

$$\pi(s,a) = \frac{x(s,a)}{\alpha(s) + \gamma \sum_{s',a'} x(s',a')\theta(s',a',s)}. \tag{5.45}$$

Clearly $\mathbf{x}$ is a solution to the π-specific Bellman flow constraints (5.41) - (5.42), and

Lemmas 5.14 and 5.15 imply that $\mathbf{x}$ is the occupancy measure of π.

For the other direction of the theorem, we assume that $\mathbf{x}$ is the occupancy measure of a stationary policy π. Lemmas 5.14 and 5.15 imply that $\mathbf{x}$ is the unique solution to the π-specific Bellman flow constraints (5.41) - (5.42). Therefore, π is given by (5.45). And since $\sum_a \pi(s,a) = 1$, we have

$$1 = \frac{\sum_a x(s,a)}{\alpha(s) + \gamma \sum_{s',a'} x(s',a')\theta(s',a',s)}$$

which can be rearranged to show that $\mathbf{x}$ satisfies the Bellman flow constraints (5.41) - (5.42), and also combined with (5.45) to show that $\pi(s,a) = \dfrac{x(s,a)}{\sum_{a'} x(s,a')}$. □

Recall that a mixed policy $\boldsymbol{\psi}$ is a distribution on the set of deterministic stationary policies Π_D, where $\psi(i)$ is the weight assigned to policy $\pi^i \in \Pi_D$. The Bellman flow constraints make it very easy to show that, for every mixed policy, there is a stationary policy that has the same value.

Theorem 5.16. *Let $\boldsymbol{\psi}$ be a mixed policy. For each $i \in \{1, \ldots, |\Pi_D|\}$, let $\mathbf{x}^i$ be the occupancy measure of $\pi^i \in \Pi_D$. Let $\tilde{\pi}$ be a stationary policy where*

$$\tilde{\pi}(s,a) = \frac{\sum_i \psi(i) x^i(s,a)}{\sum_{a'} \sum_i \psi(i) x^i(s,a')}.$$

Then $V(\tilde{\pi}) = V(\boldsymbol{\psi})$.

Proof. By Theorem 5.13, for all i, we have that $\mathbf{x}^i$ satisfies the Bellman flow constraints (5.39) - (5.40). Let $\tilde{x}(s,a) = \sum_i \psi(i) x^i(s,a)$. By linearity, $\tilde{\mathbf{x}}$ also satisfies the Bellman flow constraints. Hence, by Theorem 5.13, the stationary policy $\tilde{\pi}$ defined by $\tilde{\pi}(s,a) = \dfrac{\tilde{x}(s,a)}{\sum_{a'} \tilde{x}(s,a')}$ has occupancy measure $\tilde{\mathbf{x}}$. Therefore,

$$V(\tilde{\pi}) = \sum_{s,a} R(s)\tilde{x}(s,a) = \sum_i \psi(i) \sum_{s,a} R(s) x^i(s,a) = \sum_i \psi(i) V(\pi^i) = V(\boldsymbol{\psi})$$

where these equalities use, in order: Lemma 5.12; the definition of $\tilde{\mathbf{x}}$; Lemma 5.12; and (4.7). □

MWAL-Dual Algorithm

In this section, we make a minor modification to the MWAL algorithm so that it outputs a stationary policy instead of a mixed policy. Recall that the MWAL algorithm requires a subroutine that can compute an optimal policy in an MDP, and another subroutine that can compute that policy's feature expectations (see Lines 6 and 7 of Algorithm 5.2). Our proposal is to use the Dual LP method to find the occupancy measure $\mathbf{x}^{\pi_t}$ of the optimal policy π_t that maximizes the function $\mathbf{w}_t \cdot \boldsymbol{\mu}(\pi)$. Then, following Lemma 5.12, we compute the feature expectations $\boldsymbol{\mu}_t = \boldsymbol{\mu}(\pi_t)$ by letting $\boldsymbol{\mu}_t = \sum_{s,a} \boldsymbol{\phi}(s) x^{\pi_t}(s,a)$.

Now we can apply Theorem 5.16 to combine all the policies computed during the MWAL algorithm into a single stationary policy. This amounts to changing the last step of the MWAL algorithm (Line 11 in Algorithm 5.2) to the following:

> **Return:** The stationary policy π_A given by
>
> $$\pi_A(s,a) = \frac{\frac{1}{T}\sum_t x^{\pi_t}(s,a)}{\sum_{a'} \frac{1}{T}\sum_t x^{\pi_t}(s,a')}$$

We call this modified algorithm the MWAL-Dual algorithm, after the method it uses to compute optimal policies. It is straightforward to show that these changes to the MWAL algorithm do not affect its performance guarantee.

Theorem 5.17. *Let π_A be the stationary policy returned by the MWAL-Dual algorithm. Then*

$$V(\pi_A) - V(\pi_E) \geq v^* - O\left(\sqrt{\frac{\log k}{T(1-\gamma)^2}}\right) \tag{5.46}$$

Proof. By Theorem 5.16, the stationary policy returned by the MWAL-Dual algorithm has the same value as the mixed policy returned by the original MWAL algorithm. Hence the guarantee in (5.7) applies to the MWAL-Dual algorithm as well. □

Of course, the trick used here to convert a mixed policy to a stationary one is completely general, provided that the occupancy measures of the component policies can be computed. For example, this technique could be applied to the mixed policy output by either the Projection or Blackwell algorithm.

A significant drawback of MWAL-Dual algorithm is that it requires knowledge of the transition function θ in order to form the Dual LP. However, Wang et al. [141] describe algorithms for finding the optimal policy in an MDP that work with dual representation of policies, but are not based on linear programming. Instead, their *dual reinforcement learning algorithms* algorithms closely resembles standard reinforcement learning algorithms like value iteration, policy iteration, and Q-learning, but operate exclusively on occupancy measure. For example, in Dual Value Iteration, the occupancy measure variables are updated in each iteration according to rules that resemble the Bellman flow constraints, and convergence is proved via a contraction property. Just like the Dual LP method, the output of Dual Value Iteration is the occupancy measure $\mathbf{x}^*$ of an optimal policy π^*. Clearly, this method can be substituted for the Dual LP method in the MWAL-Dual algorithm. Importantly, some of the other algorithms due to Wang et al. [141], like the dual Q-learning, do not require any knowledge of the transition function θ.

5.6 Algorithm Based on Linear Programming

We now describe a way to use the Bellman flow constraints to find a good apprentice policy in a much more direct fashion than the MWAL algorithm. Recall the original

game-theoretic objective for apprenticeship learning:

$$\max_{\boldsymbol{\psi} \in \boldsymbol{\Psi}} \min_{\mathbf{w} \in \Delta^k} [\mathbf{w} \cdot \boldsymbol{\mu}(\boldsymbol{\psi}) - \mathbf{w} \cdot \boldsymbol{\mu}_E]$$

As we have said, the use of mixed policies in this objective is largely for algorithmic convenience. A more natural objective is to find a stationary policy that realizes

$$v^* \triangleq \max_{\boldsymbol{\pi} \in \Pi} \min_{\mathbf{w} \in \Delta^k} [\mathbf{w} \cdot \boldsymbol{\mu}(\pi) - \mathbf{w} \cdot \boldsymbol{\mu}_E] \tag{5.47}$$

where Π is the set of all stationary policies. Note that the value of v^* in (5.47) is not exactly the same as in (5.4), since here the objective has not been shifted and rescaled (as such manipulation is unnecessary for the approach used in this section).

In this section, we describe a linear program that solves (5.47). In Section 5.7, we describe experiments that show that, at least empirically, this approach is much faster than the MWAL algorithm, although it does have disadvantages, which we also illustrate in Section 5.7.

Our LPAL (Linear Programming Apprenticeship Learning) algorithm is given in Algorithm 5.4. The basic idea is to use the Bellman flow constraints (5.39) - (5.40) to define a feasible set containing all (occupancy measures of) stationary policies, and then find the best policy in that set with respect to the objective in (5.47).

Theorem 5.18. *Let π_A be the stationary policy returned by the LPAL algorithm. Then*

$$V(\pi_A) - V(\pi_E) \geq v^* - \epsilon$$

Proof. Since $\hat{\mathbf{x}}$ satisfies the Bellman flow constraints (5.39) - (5.40), by Theorem 5.13

Algorithm 5.4 LPAL algorithm

1: **Given:** MDP without a reward function, feature function $\boldsymbol{\phi}$, mentor's feature expectations $\boldsymbol{\mu}_E$.
2: Write $\boldsymbol{\phi}(s) = (\phi_1(s), \ldots, \phi_k(s))$.
3: Find an ϵ-approximate solution $(\hat{\mathbf{x}}, \hat{y})$ to this linear program:

$$\max_{\mathbf{x}, y} \quad y$$

such that

$$y \leq \sum_{s,a} \phi_i(s) x(s,a) - \mu_E(i) \text{ for all } i \in \{1, \ldots, k\}$$

$$\mathbf{x} \text{ satisfies the Bellman flow constraints (5.39) - (5.40).}$$

4: **Return:** Apprentice policy π_A defined by $\pi_A(s,a) = \dfrac{\hat{x}(s,a)}{\sum_{a'} \hat{x}(s,a')}$.

we have that $\hat{\mathbf{x}}$ is the occupancy measure of policy π_A. Now consider

$$\begin{aligned}
V(\pi_A) - V(\pi_E) &= \mathbf{w}^* \cdot \boldsymbol{\mu}(\pi_A) - \mathbf{w}^* \cdot \boldsymbol{\mu}_E && \text{(by (4.5))} \\
&= \mathbf{w}^* \cdot \sum_{s,a} \boldsymbol{\phi}(s) \hat{x}(s,a) - \mathbf{w}^* \cdot \boldsymbol{\mu}_E && \text{(Lemma 5.12)} \\
&\geq \min_{\mathbf{w} \in \Delta^k} \left[\mathbf{w} \cdot \sum_{s,a} \boldsymbol{\phi}(s) \hat{x}(s,a) - \mathbf{w} \cdot \boldsymbol{\mu}_E \right] && (\mathbf{w}^* \in \Delta^k) \\
&= \min_{i \in \{1,\ldots,k\}} \left[\sum_{s,a} \phi_i(s) \hat{x}(s,a) - \mu_E(i) \right] \\
&\geq \hat{y} && ((\hat{\mathbf{x}}, \hat{y}) \text{ is feasible})
\end{aligned}$$

It remains to show that $\hat{y} \geq v^* - \epsilon$. Since $(\hat{\mathbf{x}}, \hat{y})$ is an ϵ-approximate solution to the linear program, it suffices to show that v^* is the value of the linear program at its optimum. Let $B = \{\mathbf{x} : \mathbf{x} \text{ satisfies Bellman flow constraints}\}$. It is easy to see that the linear program in the LPAL algorithm is equivalent to

$$\max_{\mathbf{x} \in B} \min_{i \in \{1,\ldots,k\}} \left[\sum_{s,a} \phi_i(s) x(s,a) - \mu_E(i) \right] \tag{5.48}$$

We now show that v^* is equal to (5.48). Indeed

$$\begin{aligned} v^* &= \max_{\boldsymbol{\pi} \in \Pi} \min_{\mathbf{w} \in \Delta^k} [\mathbf{w} \cdot \boldsymbol{\mu}(\pi) - \mathbf{w} \cdot \boldsymbol{\mu}_E] && \text{(by (5.47))} \\ &= \max_{\mathbf{x} \in B} \min_{\mathbf{w} \in \Delta^k} \left[\mathbf{w} \cdot \sum_{s,a} \boldsymbol{\phi}(s) x(s,a) - \mathbf{w} \cdot \boldsymbol{\mu}_E \right] && \text{(Lemma 5.12 and Theorem 5.13)} \\ &= \max_{\mathbf{x} \in B} \min_{i \in \{1,\ldots,k\}} \left[\sum_{s,a} \phi_i(s) x(s,a) - \mu_E(i) \right] \end{aligned}$$

□

Theorem 5.18 does not actually provide a bound on the running time of the algorithm, since it does not bound the time required to produce an ϵ-approximate solution to a linear program. For a typical LP solver, the running time for a linear program with $O(n)$ variables and constraints is $O(n^{3.5} \log n \log(1/\epsilon))$ [24]. For the linear program in the LPAL algorithm $n = O(|\mathcal{S}||\mathcal{A}| + k)$.

5.7 Experiments

Our experiments in this section are designed to highlight the advantages of the algorithms we developed in this chapter, as well as illustrate the consequences of some of our theoretical findings. In Section 5.7.1, we give examples where the MWAL algorithm learns an apprentice policy that outperforms the mentor, unlike earlier algorithms that mimic the mentor. In Section 5.7.2, we show that the LPAL algorithm often learns an apprentice policy in significantly less time than the MWAL algorithm. In Section 5.7.2, we explain that this improvement in running time comes at a price: For many suboptimal mentor policies, the policy learned by LPAL is much worse than the policy learned by MWAL, even though they both have the same theoretical guarantee. This phenomenon may be at least partly explained by our results about lexicographic optimality.

5.7.1 MWAL Policy Better Than Mentor Policy

Recall our discussion in Section 5.2.1, where we observed that the bounds for apprenticeship learning that mimic the mentor are "two-sided", while the bound for the MWAL algorithm is "one-sided". In this section, we present experiments that illustrate the consequences of this difference.

We tested the MWAL algorithm and the projection algorithm on the small car driving simulator described in Section 4.1. A similar simulator was used by Abbeel and Ng [1]. In this simulator, the apprentice must navigate a car through randomly-generated traffic on a three-lane highway. We define three features for this environment: a collision feature (0 if contact with another car, and 1/2 otherwise), an off-road feature (0 if off-road, and 1/2 otherwise), and a speed feature (1/2, 3/4 and 1 for each of the three possible speeds, with higher values corresponding to higher speeds). Note that the features encode that, other things being equal, speed is good, and collisions and off-roads are bad.

	Fast Mentor	Proj	MWAL		Bad Mentor	Proj.	MWAL		No Mentor	MWAL
Speed	Fast	Fast	Fast		Slow	Slow	Medium		-	Medium
Collisions (per sec)	1.1	1.1	0.5		2.23	2.23	0		-	0
Off-roads (per sec)	0	0	0		8.0	8.0	0		-	0

Table 5.1: MWAL vs. Projection algorithm.

Table 5.1 displays the results of using the MWAL and projection algorithms to learn a driving policy by observing two kinds of mentors: a "fast" mentor and a "bad" mentor.

The "fast" mentor's policy is to drive at the top speed, while being indifferent to colliding with other cars or going off-rood. As expected, the projection algorithm's policy mimics this behavior. The MWAL algorithm's policy also matches the speed of the mentor, but subject to this constraint stays on the road and avoids collisons as best as possible. The MWAL algorithm's policy does not slow down to avoid all collisions because, in the worst-case, the true weight vector $\mathbf{w}^*$ may be adversarially

set to assign all weight to the speed feature, in which case any policy that does not drive at the fastest speed would underperform the mentor.

The "bad" mentor's policy illustrates another situation in which the MWAL algorithm produces a significantly better policy than the projection algorithm. In this experiment, the mentor's policy is to drive at the slowest speed, and to delibrately hit other cars and veer off-road. This is essentially the worst possible policy, regardless of the value of the actual weight vector $\mathbf{w}^*$. And while the projection algorithm imitates this behavior, the MWAL algorithm instead follows a conservative policy that effectively ignores the mentor, since it is able to conclude that the conservative policy dominates the behavior exhibited by the mentor.

We also applied the MWAL algorithm in a setting in which there is no mentor at all (see Section 5.2.2). The resulting policy is one in which the apprentice drives as fast as possible without hitting any other cars or veering off-road. This is the best policy for the worst-case possibility for $\mathbf{w}^*$. Indeed, if the apprentice were to drive any faster, it would be impossible to avoid every other car with certainty, and then $\mathbf{w}^*$ could be adversarially set to assign all weight to the collision feature, resulting in a lower value for the apprentice's policy.

5.7.2 LPAL Converges Faster Than MWAL

To compare the running time of the MWAL, MWAL-Dual, and LPAL algorithms, we tested each algorithm in several gridworld environments. Each gridworld was an $N \times N$ square of states. Movement was possible in the four compass directions, and each action had a 30% chance of causing a transition to a random state. Each gridworld was partitioned into several square regions, each of size $M \times M$. We always chose M so that it evenly divided N, so that each gridworld had $k = (\frac{N}{M})^2$ regions. Each gridworld also had k features, represented by the feature vector $\boldsymbol{\phi}$, where the ith feature was a 0-1 indicator function for the ith region. Similar gridworld environments

were used by Abbeel and Ng [1].

For each gridworld, in each trial, we randomly chose a sparse weight vector w^*. Recall that the true reward function has the form $R(s) = \mathbf{w}^* \cdot \boldsymbol{\phi}(s)$, so in these experiments the true reward function encoded that some regions are more desirable than others. In each trial, we let the mentor policy π_E be the optimal policy with respect to R, and then supplied the feature expectations vector $\boldsymbol{\mu}(\pi_E)$ to the MWAL, MWAL-Dual and LPAL algorithms.[5]

Our experiments were run on an ordinary desktop computer. We used the Matlab-based `cvx` package [40] for our LP solver. Each of the values in the tables below is the time, in seconds, that the algorithm took to find an apprentice policy π_A such that $V(\pi_A) \geq 0.95V(\pi_E)$. Each running time is the average of 10 trials.

Table 5.2: Time (sec) to find π_A s. t. $V(\pi_A) \geq 0.95V(\pi_E)$

Gridworld Size	MWAL (sec)	MWAL-Dual (sec)	LPAL (sec)
16×16	6.43	46.99	1.46
24×24	14.45	90.16	1.55
32×32	27.23	247.38	2.76
48×48	61.37	791.61	8.62
64×64	114.12	3651.70	30.52
128×128	406.24	4952.74	80.21
256×256	1873.93	29988.85	588.60

Table 5.3: Time (sec) to find π_A s. t. $V(\pi_A) \geq 0.95V(\pi_E)$

Gridworld Size	Number of Regions	MWAL (sec)	MWAL-Dual (sec)	LPAL (sec)
24×24	64	14.45	90.16	1.55
	144	32.33	97.58	2.64
	576	129.87	120.82	1.86
32×32	64	27.23	247.38	2.76
	256	107.11	270.71	8.43
	1024	440.64	361.36	4.75
48×48	64	61.37	791.61	8.62
	144	135.83	800.23	11.42
	256	244.46	815.66	16.89
	576	575.34	847.38	16.33
	2304	2320.71	1128.32	11.14

[5] In practice, π_E will be unknown, and so the feature expectations would need to be estimated from a data set of mentor sample trajectories. However, since we are primarily concerned with computational complexity in this section, and not sample complexity, we sidestep this issue and just compute each $\boldsymbol{\mu}(\pi_E)$ directly.

In the first set of experiments (Table 5.2), we tested the algorithms in gridworlds of varying sizes, while keeping the number of regions in each gridworld fixed (64 regions). Recall that the number of regions is equal to the number of features. In the next set of experiments (Table 5.3), we varied the number of regions while keeping the size of the gridworld fixed.

Several remarks about these results are in order. For every gridworld size and every number of regions, the LPAL algorithm is substantially faster than the other algorithms — in some cases two orders of magnitude faster. As we previously noted, LP solvers are often much more efficient than their theoretical guarantees. Interestingly, in Table 5.3, the running time for LPAL eventually decreases as the number of regions increases. This may be because the number of constraints in the linear program increases with the number of regions, and more constraints often make a linear program problem easier to solve.

Also, the MWAL-Dual algorithm is much slower than the other algorithms. We suspect this is only because the MWAL-Dual algorithm calls the LP solver in each iteration (unlike the LPAL algorithm, which calls it just once), and there is substantial overhead to doing this. Modifying MWAL-Dual so that it uses the LP solver as less of a blackbox may be a way to alleviate this problem.

5.7.3 Suboptimal Mentors

In light of the results from the previous section, one might reasonably wonder whether there is any argument for using an algorithm other than LPAL. Recall that, in those experiments, the mentor's policy was an optimal policy for the unknown reward function. In this section we explore the behavior of each algorithm when this is not the case, and find that MWAL produces better apprentice policies than LPAL. Our experiments were run on the car driving simulator from Section 4.1.

We designed three mentors for these experiments, described in Table 5.4. Each

mentor is optimal for one of the features, and mediocre for the other two. Therefore each mentor policy π_E is an optimal policy if $\mathbf{w}^* = \mathbf{w}_E$, where $\mathbf{w}_E$ is the weight vector that places all weight on the feature for which π_E is optimal. At the same time, each π_E is very likely to be suboptimal for a randomly chosen $\mathbf{w}^*$.

We used the MWAL and LPAL algorithms to learn apprentice policies from each of these mentors. The results are presented in Table 5.5. We let $\gamma = 0.9$, so the maximum value of the feature corresponding to speed was 10, and for the others it was 5. Each of the reported policy values for randomly chosen $\mathbf{w}^*$ was averaged over 10,000 uniformly sampled $\mathbf{w}^*$'s.

Table 5.4: Mentor types

	Speed	Collisions (per sec)	Off-roads (per sec)
"Fast" Mentor	Fast	1.1	10
"Avoid" Mentor	Slow	0	10
"Road" Mentor	Slow	1.1	0

Table 5.5: Driving simulator experiments.

Mentor type	Algorithm used	$\mathbf{w}^* = \mathbf{w}_E$		$\mathbf{w}^*$ chosen randomly	
		$V(\pi_A)$	$V(\pi_E)$	$V(\pi_A)$	$V(\pi_E)$
"Fast"	MWAL	10	10	9.83	8.25
	LPAL	10	10	8.84	8.25
"Avoid"	MWAL	5	5	8.76	6.32
	LPAL	5	5	7.26	6.32
"Road"	MWAL	5	5	9.74	7.49
	LPAL	5	5	8.12	7.49

Notice that for each mentor, when $\mathbf{w}^*$ is chosen randomly, MWAL outputs better apprentice policies than LPAL. That is, although the theoretical performance guarantees of both the MWAL and LPAL algorithm are identical, the results in Table 5.5 suggest that the two algorithms are not equally effective. Only the MWAL algorithm enjoys the lexicographic properties that we discussed in Chapter 3, and we speculate that this accounts for the difference.

5.8 Other Related Work

One way to view apprenticeship learning is as a generalization of reinforcement learning, where the goal is to find an optimal policy with respect to several reward functions (i.e., the features), instead of a single reward function. This is essentially the goal of *multicriteria MDPs*, also known as multiobjective MDPs or constrained MDPs. Frid (1972) and Kallenberg (1981) were among the first to study multicriteria MDPs, and more recent work has focused on proving the existence of optimal policies, characterizing their structure, and describing algorithms for computing them. Feinberg and Shwartz [26], Feinberg and Shwartz [27], Dolgov and Durfee [21] and Wiering and de Jong [144] offer typical examples. More generally, Barrett and Narayanan [6] described an algorithm that computes the set of *all* policies that are optimal for *at least one* of the reward functions. However, their algorithm has a running time that is exponential in the number of reward functions.

Unlike the algorithms in this chapter, nearly all existing algorithms for multicriteria MDPs and its variants assume that the transition function is given. One exception is the algorithm given by Gábor et al. [33], who described an approach in which there is a fixed preference order on the reward functions (see also Natarajan and Tadepalli [87], who allowed the relative importance of the various reward functions to change over time).

Several authors have previously studied reinforcement learning in settings where the reward function is selected by an adversary. Perhaps the earliest work that can be placed in this category is minimax Q-learning [72] (see also a proof of its convergence due to Littman and Szepesvri [73]).

Even-Dar et al. [23] studied a reinforcement learning problem where the adversarially chosen reward function is allowed to vary every time step. They derived an efficient algorithm that uses the MW algorithm as a subroutine, and that earns nearly as much average reward as the best stationary policy. However, unlike in

apprenticeship learning, the reward function is revealed to the algorithm after every time step.

McMahan et al. [77] took a very similar approach as we have done in this chapter: a single unknown reward function is selected by an adversary from a finite set of possible reward functions. They also formulated the problem as two-player zero-sum game, and their *double oracle* algorithm resembles a hybrid of the linear programming and fictitious play approaches to solving a game. In each round of the algorithm, each player selects a best response to an optimal strategy from the subset of the strategies previously chosen by the other player. Theoretically, the weakness of this algorithm it is only guaranteed to converge in time proportional to the number of policies, i.e., exponential time. Moreover, the algorithm does not enjoy any of the lexicographic convergence properties of the MWAL algorithm. Incidentally, Zinkevich et al. [151] have observed that the double oracle algorithm is closely related to earlier work by Dantzig and Wolfe [18], Gilmore and Gomory [37] and Lemke and J. T. Howson [68].

5.9 Conclusion

In this chapter, we posed the problem of apprenticeship learning as a two-person zero-sum game, in which the apprentice chooses a policy, and the environment adversarially chooses a reward function. We derived a multiplicative weights algorithm for this formulation, with the key property that it can leverage prior beliefs about the relationship between the features and the reward function. As a result, the algorithm can produce a policy that is significantly better than the mentor's policy with respect to the unknown reward function, while at the same time is guaranteed to be no worse. The simplest version of the algorithm outputs a mixed policy, a type of policy that has an unnatural structure. We presented several algorithms that output stationary policies instead, one of which is based on linear programming and is experimentallly

much faster.

Chapter 6

Imitation Learning with a Value-Based Prior

In Chapters 4 and 5, we argued that specifying a reward function in a reinforcement learning problem can sometimes be difficult, and we dealt with that problem by assuming that the rewards are unknown, and possibly even chosen by an adversary. Our algorithms in Chapters 4 and 5 were designed within the apprenticeship learning framework, where all information about the unknown rewards is obtained indirectly from demonstrations by a mentor.

In this chapter, we will take a different approach to the problem of dealing with a hard-to-specify reward function. Instead of assuming that the rewards are unknown, we assume that the we are given a complete reward function that encodes *prior knowledge* about the mentor's behavior. And instead of learning a behavior that maximizes rewards, our goal will be to *imitate* the mentor, and the rewards are only used to *bias* the estimation of her behavior. Since this goal is quite different than the goal in previous chapters, we call the learning agent in this scenario the *imitator* instead of the apprentice.

In a way, our approach in this chapter is the reverse of the approach used in

Chapters 4 and 5. In those chapters, the correct behavior was defined in terms of rewards, and the mentor's demonstrations were used as a "hint" about what the correct behavior was. In this chapter, the goal will be to imitate the mentor's behavior, and the rewards will be used a "hint".

More precisely, our approach in this chapter can be viewed as using a *value-based prior* to learn a mentor's policy, meaning that we use the reward function in an MDP to encode the imitator's prior belief about the mentor's policy. We assume that the prior probability of any policy being the mentor's increases with the value of that policy in the MDP. In this way, instead of relying solely on rewards or soley on evidence, the imitator smoothly integrates both prior knowledge and observed information about the mentor's policy.

This approach is motivated by the problem of *dialog management* in human-computer spoken dialog systems. In Section 6.1 we review this problem in greater detail, and explain why the use of a value-based prior is a suitable approach to solving it. In Sections 6.2 and 6.3 we describe how a value-based prior can be expressed compactly using Bellman's equations, and in Section 6.4 we present an efficient algorithm for maximizing the resulting posterior distribution.

Section 6.5 presents experimental results on a synthetic domain that show that, compared to existing methods, a value-based prior substantially speeds the estimation of the mentor's policy. Beginning in Section 6.6.5, we apply our algorithm to a spoken dialog system currently in use in a large corporation. This application has an additional complication: The observations of the mentor's behavior are corrupted by noise. To handle this noise, we show how our algorithm can be used as the "M-step" of a variant of the well-known EM algorithm [19]. The material in this chapter was described by Syed and Schapire [126] and Syed and Williams [129].

6.1 Motivation

A *dialog manager* is a program that controls the actions of an automated telephone agent, also known as a *spoken dialog system*, such as the kind one encounters when calling a company's customer service number. Instead of asking the caller to navigate menus by pressing buttons, these agents encourage customers to speak freely, and attempt to offer an experience comparable to that of speaking to a live operator. The dialog manager makes decisions about which questions to ask, how to deal with unexpected responses, what to do when the customer is misunderstood (ask them to clarify? make a best guess and move on?), and when to give up and transfer the customer to a human operator.

Reinforcement learning algorithms have been used to train dialog managers by interacting with users [69, 120]. However, this approach requires the assertion of a reward function that is based largely on intuition, since customers rarely give a clear indication about whether they are satisfied with a dialog. Indeed, Walker et al. [140] showed that evaluating the performance of a dialog manager is itself a challenging task, which calls into question whether reinforcement learning is sufficient to solve this problem, and suggests that some form of imitation may be needed.

Another challenge is the scarcity of suitable training opportunities, since new dialog management strategies cannot be tested on a static corpus. They have to be tried in real dialogs with actual users, which is, needless to say, an expensive proposition. As a result, there has been much interest in building user models, i.e., simulators that imitate the behavior of customers; Schatzmann et al. [114] provide a survey and comparison of some attempts at learning user models from data. A common theme in this work has been to leverage prior knowledge, and restrict the space of models to those that encode realistic user behavior, in the hope that less data will be needed for training.

The value-based prior presented in this chapter has been developed with these

issues in mind. At the same time, the framework and algorithms presented here are intended to be completely general, and not specific to dialog management. We assume that an imitator is observing a mentor acting in a stochastic environment, and that the imitator wants to estimate a model of the mentor's behavior. We further assume that the mentor is behaving in a roughly reward-seeking manner. The imitator uses the value function of an MDP to help guide its estimate towards the correct policy. For example, in the domain of dialog management, we can assign higher rewards in the MDP to states that are closer to the end of the conversation. In this way, we can leverage our knowledge that customers and operators are both trying to complete their conversations as soon as possible, without needing to specify exactly how they are trying to accomplish that goal.

6.2 Problem Formulation

We assume that the imitator is given a finite-horizon MDP, which we call the *modeling MDP*. We chose a finite horizon because spoken dialogs typically occur in episodes. Recall from Section 4.5 that a finite-horizon MDP consists of a finite set of states $\mathcal{S}$, a finite set of actions $\mathcal{A}$, a horizon H, a reward function $R : \mathcal{S} \to \mathbb{R}$, an initial state distribution $\boldsymbol{\alpha}$, and a transition function $\boldsymbol{\theta}$ (the assumption that the transition function is given can be relaxed; see Section 6.4.3). It is important to note that it is *not* the imitator's objective to compute an optimal policy for the modeling MDP. Rather, the goal is to estimate the mentor's policy, and the modeling MDP is used to encode the imitator's prior beliefs about that policy.

We further assume that we are given a data set $\mathcal{X}$ of state-action trajectories of the mentor acting in this environment. Each trajectory is a sequence of H state-action pairs, i.e., $(s_1^i, a_1^i), \ldots (s_H^i, a_H^i)$. Our objective is to estimate the policy $\boldsymbol{\pi}$ that governs the mentor's behavior, where π_{sa}^t is the probability the mentor takes action

a in state s at time t. The *maximum a posteriori* (MAP) estimate for the mentor's policy is given by

$$\begin{aligned}\hat{\boldsymbol{\pi}} &= \arg\max_{\boldsymbol{\pi}} \log P(\mathcal{X} \mid \boldsymbol{\pi}) + \log P(\boldsymbol{\pi}) \\ &= \arg\max_{\boldsymbol{\pi}} \sum_{s,a,t} K_{sat} \log \pi^t_{sa} + \log P(\boldsymbol{\pi}),\end{aligned}$$

where K_{sat} is the number of times in $\mathcal{X}$ that action a is taken in state s at time t. If the prior distribution $P(\boldsymbol{\pi})$ is uniform, then $\hat{\boldsymbol{\pi}}$ can be calculated analytically; the solution is just $\hat{\pi}^t_{sa} = \dfrac{K_{sat}}{\sum_{a'} K_{sa't}}$.

In this chapter, we show how to assert a prior distribution $P(\boldsymbol{\pi})$ that gives greater weight to policies that have greater value in the modeling MDP. As usual, the value of policy $\boldsymbol{\pi}$ is defined

$$V(\boldsymbol{\pi}) = E\left[\sum_{t=1}^{H} R(s_t) \;\middle|\; \boldsymbol{\pi}, \boldsymbol{\theta}, \boldsymbol{\alpha}\right]$$

If we let $P(\boldsymbol{\pi}) = \exp(\kappa V(\boldsymbol{\pi}))$, then the MAP estimate is now given by

$$\begin{aligned}\hat{\boldsymbol{\pi}} &= \arg\max_{\boldsymbol{\pi}} \sum_{s,a,t} K_{sat} \log \pi^t_{sa} + \kappa V(\boldsymbol{\pi}) \qquad (6.1) \\ &\triangleq \arg\max_{\boldsymbol{\pi}} L(\boldsymbol{\pi}).\end{aligned}$$

Here, κ can be viewed as a trade-off parameter that determines how much relative weight $P(\boldsymbol{\pi})$ assigns to high-value policies. Also note that $P(\boldsymbol{\pi})$ in this case is an unnormalized prior, as it does not necessarily intergrate to 1, and so (6.1) is perhaps more appropriately termed the estimate which maximizes a *penalized likelihood* [41].

6.3 Representing the Value-Based Prior

The trouble with finding the maximum of $L(\boldsymbol{\pi})$ directly is that it is inefficient to represent the expression for $V(\boldsymbol{\pi})$ directly. This is because

$$\begin{aligned} & V(\boldsymbol{\pi}) \\ = & \sum_s \alpha_s V_s^1 \\ = & \sum_s \alpha_s (R_s + \sum_{a,s'} \pi_{sa}^1 \theta_{sas'} V_{s'}^2) \\ = & \sum_s \alpha_s (R_s + \sum_{a,s'} \pi_{sa}^1 \theta_{sas'} (R_{s'} + \sum_{a,s''} \pi_{s'a}^2 \theta_{s'as''} V_{s''}^3)) \\ = & \ \ldots \end{aligned}$$

It is easy to see that this quickly gets out of hand. If we fully expand the definition of $V(\boldsymbol{\pi})$, the final expression will contain N^H terms, which is far too many to explicitly represent. We can express $V(\boldsymbol{\pi})$ more compactly by using Bellman's equations (see Section 4.2.1), which yields the following optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\pi}, \mathbf{V}} \quad & \sum_{s,a,t} K_{sat} \log \pi_{sa}^t + \kappa \sum_s \alpha_s V_s^1 \\ \text{subject to:} & \\ \forall s,\ \forall\, t < H \quad & V_s^t = R(s) + \sum_{a,s'} \pi_{sa}^t \theta_{sas'}^t V_{s'}^{t+1} \qquad (6.2) \\ \forall s \quad & V_s^H = R(s) \\ \forall s, t \quad & \sum_a \pi_{sa}^t = 1 \\ \forall s, a, t \quad & \pi_{sa}^t \geq 0 \end{aligned}$$

where V_s^t is the value of the policy in state s at time t, and $\mathbf{V}$ is the vector of all these values. This problem is still difficult, however, since it involves nonconvex constraints — note that Bellman's equations (6.2) are bilinear in $\boldsymbol{\pi}$ and $\mathbf{V}$. To circumvent this,

we will perform an *alternating maximization* instead. Our algorithm is described in the next section.

6.4 Algorithm and Analysis

In this section, we present an iterative algorithm that converges to a stationary point $L(\boldsymbol{\pi})$, the function in Equation (6.1). In Section 6.4.1 we provide a detailed description of each iteration of the algorithm, and in Section 6.4.2 we give a proof of its convergence. In Section 6.4.3 we show how the algorithm can be extended to a setting where both the mentor policy and the transition function are unknown.

Our iterative algorithm for maximizing $L(\boldsymbol{\pi})$ will optimize certain groups of the variables at a time, while leaving the other variables fixed. Let $\boldsymbol{\pi} = \left(\boldsymbol{\pi}^1, \ldots, \boldsymbol{\pi}^H\right)$ and $\mathbf{V} = \left(\mathbf{V}^1, \ldots, \mathbf{V}^H\right)$. We will maximize $L(\boldsymbol{\pi})$ over just $\boldsymbol{\pi}^1$, then $\boldsymbol{\pi}^1$, and so on until $\boldsymbol{\pi}^H$, and then repeat the cycle until convergence (see Algorithm 6.1). In the iteration for $\boldsymbol{\pi}^\tau$, the values for $\boldsymbol{\pi}^1, \ldots, \boldsymbol{\pi}^{\tau-1}, \boldsymbol{\pi}^{\tau+1}, \ldots, \boldsymbol{\pi}^H$ are carried over from previous iterations and are held fixed while $\boldsymbol{\pi}^\tau$ is optimized. Taking this alternating approach has the effect of linearizing the constraints in (6.2), since $\mathbf{V}^{\tau+1}$, $\mathbf{V}^{\tau+2}$, $\ldots$, $\mathbf{V}^H$ are not affected by changes to $\boldsymbol{\pi}^\tau$, and therefore can also be held fixed without impacting the maximization over $\boldsymbol{\pi}^\tau$.

Due to the linearization of the constraints in (6.2), each iteration of Algorithm 6.1 is just a convex optimization problem, and hence can be solved by any of a number of standard techniques, such as interior point methods. However, general-purpose methods are quite complex; fortunately they turn out to be unnecessary in this case. In Section 6.4.1, we describe a relatively simple procedure that solves this particular optimization problem in $O(|\mathcal{S}|^2|\mathcal{A}|H + |\mathcal{S}||\mathcal{A}|(\log|\mathcal{A}| + \log|\mathcal{X}|))$ time.

Algorithm 6.1 Find a stationary point of the log posterior.

Let $\boldsymbol{\pi} = \left(\boldsymbol{\pi}^1, \ldots, \boldsymbol{\pi}^H\right)$.
Let $L(\boldsymbol{\pi}) = \sum_{s,a,t} K_{sat} \log \pi^t_{sa} + \kappa V(\boldsymbol{\pi})$.
Initialize $\tilde{\boldsymbol{\pi}}$ to any point in the interior of the set of all policies Π.
$\tau \leftarrow 1$.
repeat
  $\boldsymbol{\pi} \leftarrow \tilde{\boldsymbol{\pi}}$
  $\tilde{\boldsymbol{\pi}}^\tau = \arg\max_{\boldsymbol{\pi}^\tau} L(\boldsymbol{\pi})$
  $\tilde{\boldsymbol{\pi}} = \left(\boldsymbol{\pi}^1, \ldots, \boldsymbol{\pi}^{\tau-1}, \tilde{\boldsymbol{\pi}}^\tau, \boldsymbol{\pi}^{\tau+1}, \ldots, \boldsymbol{\pi}^H\right)$
  if $\tau = H$ **then**
    $\tau \leftarrow 1$
  else
    $\tau \leftarrow \tau + 1$
  end if
until convergence

6.4.1 Optimization Procedure

In each iteration of Algorithm 6.1, we maximize $L(\boldsymbol{\pi})$ over $\boldsymbol{\pi}^\tau$, for some $\tau \in \{1, \ldots, H\}$. When $\tau \neq H$, the corresponding convex optimization (after dropping constant terms) is[1]

$$\begin{aligned}
\max_{\boldsymbol{\pi}^\tau, \mathbf{V}^1, \ldots, \mathbf{V}^\tau} \quad & \sum_{s,a} K_{sa\tau} \log \pi^\tau_{sa} + \kappa \sum_s \alpha_s V^1_s \\
\text{subject to:} & \\
\forall s,\ \forall\ t \leq \tau \quad & V^t_s = R(s) + \sum_{a,s'} \pi^t_{sa} \theta^t_{sas'} V^{t+1}_{s'} \\
\forall s \quad & \sum_a \pi^\tau_{sa} = 1 \\
\forall s, a \quad & \pi^\tau_{sa} \geq 0.
\end{aligned}$$

Recall that $\boldsymbol{\pi}^1, \ldots, \boldsymbol{\pi}^{\tau-1}, \boldsymbol{\pi}^{\tau+1}, \ldots, \boldsymbol{\pi}^H$ and $\mathbf{V}^{\tau+1}, \ldots, \mathbf{V}^H$ are constants in this problem; their values are carried over from previous iterations.

To solve the optimization, we need to find a solution to the *KKT conditions* [12],

[1] The solution for the $\tau = H$ case is similar to the procedure described in this section, except it is even simpler, so we omit its discussion.

i.e., a solution $(\boldsymbol{\pi}^\tau, \mathbf{V}^1, \ldots, \mathbf{V}^\tau, \boldsymbol{\lambda})$ that is both feasible and also satifies

$$\begin{aligned} & \nabla \mathcal{L}\left(\boldsymbol{\pi}^\tau, \mathbf{V}^1, \ldots, \mathbf{V}^\tau, \boldsymbol{\lambda}\right) = 0 \\ \forall s, a \quad & \lambda^\pi_{sa} \geq 0 \\ \forall s, a \quad & \lambda^\pi_{sa} \cdot \pi^\tau_{sa} = 0 \end{aligned}$$

where $\boldsymbol{\lambda} = \{\lambda^V_{st}, \lambda^\pi_s, \lambda^\pi_{sa} \mid s \in \mathcal{S}, a \in \mathcal{A}, t \leq \tau\}$, the Lagrangian $\mathcal{L}\left(\boldsymbol{\pi}^\tau, \mathbf{V}^1, \ldots, \mathbf{V}^\tau, \boldsymbol{\lambda}\right)$ is given by

$$\begin{aligned} \mathcal{L}\left(\boldsymbol{\pi}^\tau, \mathbf{V}^1, \ldots, \mathbf{V}^\tau, \boldsymbol{\lambda}\right) = & \sum_{s,a} K_{sa\tau} \log \pi^\tau_{sa} + \kappa \sum_s \alpha_s V^1_s \\ & + \sum_{\substack{s \\ t \leq \tau}} \lambda^V_{st} \left[R_s + \sum_{a,s'} \pi^t_{sa} \theta^t_{sas'} V^{t+1}_{s'} - V^t_s \right] \\ & + \sum_s \lambda^\pi_s \left[1 - \sum_a \pi^\tau_{sa} \right] + \sum_{s,a} \lambda^\pi_{sa} \cdot \pi^\tau_{sa} \end{aligned}$$

and the gradient of $\mathcal{L}$ is taken with respect to $(\boldsymbol{\pi}^\tau, \mathbf{V}^1, \ldots, \mathbf{V}^\tau)$.

Below we outline a three-step procedure for finding $(\boldsymbol{\pi}^\tau, \mathbf{V}^1, \ldots, \mathbf{V}^\tau, \boldsymbol{\lambda})$ that satisfies the KKT conditions.

Step 1: Find the λ^V_{st}'s

From the KKT conditions, we must have that

$$\frac{\partial \mathcal{L}}{\partial V^t_s} = 0 \quad \forall s, \ \forall t \leq \tau.$$

This yields

$$\begin{aligned}
\lambda^V_{s1} &= \kappa\alpha_s \\
\lambda^V_{st} &= \sum_{s',a} \lambda^V_{s't-1}\pi^{t-1}_{s'a}\theta^t_{s'as} \quad \text{for } 1 < t \leq \tau
\end{aligned}$$

which allows us to inductively compute all the λ^V_{st}'s. We can see from this expression that

$$\lambda^V_{st} = \kappa \Pr[s_t = s \mid \boldsymbol{\pi}, \boldsymbol{\theta}, \boldsymbol{\alpha}]$$

i.e., λ^V_{st} is equal to the occupancy measure of state s at time t under policy $\boldsymbol{\pi}$, but scaled by κ.

Step 2: Find the λ^π_s's, λ^π_{sa}'s and π^τ_{sa}'s

To simplify notation, define

$$\begin{aligned}
B_{sa\tau} &\triangleq \lambda^V_{s\tau} \sum_{s'} \theta^\tau_{sas'} V^{\tau+1}_{s'} \\
\mathcal{A}^0_s &\triangleq \{a \in \mathcal{A} \mid K_{sa\tau} = 0\} \\
\mathcal{A}^{\neg 0}_s &\triangleq \mathcal{A} \setminus \mathcal{A}^0_s.
\end{aligned}$$

Let us focus on a particular state s. We know that $\sum_a \pi^\tau_{sa} = 1$ and $\pi^\tau_{sa} \geq 0$ for all a. Suppose we can find a value of λ^π_s such that

$$\sum_{a \in \mathcal{A}^{\neg 0}_s} \frac{K_{sa\tau}}{\lambda^\pi_s - B_{sa\tau}} = 1 \tag{6.3}$$

$$\frac{K_{sa\tau}}{\lambda^\pi_s - B_{sa\tau}} \geq 0 \quad \forall a \in \mathcal{A}^{\neg 0}_s. \tag{6.4}$$

If it happens that $\lambda^\pi_s \geq \max_{a \in \mathcal{A}} B_{sa\tau}$, then we can satisfy all the relevant KKT

conditions by setting

$$\begin{aligned}
\lambda^{\pi}_{sa} &= 0 && \forall a \in \mathcal{A}^{\neg 0}_s \\
\lambda^{\pi}_{sa} &= \lambda^{\pi}_s - B_{sa\tau} && \forall a \in \mathcal{A}^{0}_s \\
\pi^{\tau}_{sa} &= \frac{K_{sa\tau}}{\lambda^{\pi}_s - B_{sa\tau}} && \forall a \in \mathcal{A}^{\neg 0}_s \\
\pi^{\tau}_{sa} &= 0 && \forall a \in \mathcal{A}^{0}_s.
\end{aligned}$$

On the other hand, if $\lambda^{\pi}_s < \max_{a \in \mathcal{A}} \ B_{sa\tau}$ for the value of λ^{π}_s that solves (6.3) and (6.4), then we can satisfy the relevant KKT conditions by first letting $\lambda^{\pi}_s = \max_{a \in \mathcal{A}} \ B_{sa\tau}$, and then setting

$$\begin{aligned}
\lambda^{\pi}_{sa} &= 0 && \forall a \in \mathcal{A}^{\neg 0}_s \\
\lambda^{\pi}_{sa} &= \lambda^{\pi}_s - B_{sa\tau} && \forall a \in \mathcal{A}^{0}_s \\
\pi^{\tau}_{sa} &= \frac{K_{sa\tau}}{\lambda^{\pi}_s - B_{sa\tau}} && \forall a \in \mathcal{A}^{\neg 0}_s \\
\pi^{\tau}_{sa} &= 0 && \forall a \in \mathcal{A}^{0}_s \setminus \{a^*\} \\
\pi^{\tau}_{sa^*} &= 1 - \textstyle\sum_{a \in \mathcal{A}^{\neg 0}_s} \pi^{\tau}_{sa}.
\end{aligned}$$

where $a^* = \arg\max_{a \in \mathcal{A}} \ B_{sa\tau}$.

So it remains to show that we can easily find a λ^{π}_s that solves (6.3) and (6.4). Define

$$\begin{aligned}
B_{\max} &\triangleq \max_{a \in \mathcal{A}^{\neg 0}_s} \ B_{sa\tau} \\
K_{\max} &\triangleq \max_{a \in \mathcal{A}^{\neg 0}_s} \ K_{sa\tau} \\
K_{\min} &\triangleq \min_{a \in \mathcal{A}^{\neg 0}_s} \ K_{sa\tau}
\end{aligned}$$

and observe that

$$\begin{aligned} \lambda_s^\pi &= K_{\min} + B_{\max} \\ &\Rightarrow \sum_a \frac{K_{sa\tau}}{\lambda_s^\pi - B_{sa\tau}} \geq 1 \end{aligned}$$

and

$$\begin{aligned} \lambda_s^\pi &= |\mathcal{A}| \cdot K_{\max} + B_{\max} \\ &\Rightarrow \sum_a \frac{K_{sa\tau}}{\lambda_s^\pi - B_{sa\tau}} \leq 1. \end{aligned}$$

Moreover, the left-hand side of (6.3) is strictly monotone in λ_s^π, and $\lambda_s^\pi \in [K_{\min} + B_{\max},\ |\mathcal{A}| \cdot K_{\max} + B_{\max}]$ satisfies (6.4). Putting all this together with the Intermediate Value Theorem [110], we conclude that there exists a unique $\lambda_s^\pi \in [K_{\min} + B_{\max},\ |\mathcal{A}| \cdot K_{\max} + B_{\max}]$ that satisfies (6.3) and (6.4), so we can use a simple root-finding algorithm such as the bisection method to approximate it within a constant ϵ.

Step 3: Find the V_s^t's

Since we know the π_{sa}^τ's now, all the $V_s^1, \ldots, V_s^\tau$'s can be computed inductively.

$$V_s^t = R(s) + \sum_{a,s'} \pi_{sa}^t \theta_{sas'}^t V_{s'}^{t+1} \quad \forall s \in \mathcal{S},\ \forall\ t \leq \tau.$$

Running time

Recall that $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces, respectively, $\mathcal{X}$ is the data set of state-action trajectories, H is the length of the horizon, and ϵ is the approximation error of the root-finding algorithm used in Step 2.

Steps 1 and 3 both take $O(|\mathcal{S}|^2|\mathcal{A}|H)$ time, and step 2 takes $O(|\mathcal{S}||\mathcal{A}|(\log|\mathcal{A}| +$

$\log|\mathcal{X}| + \log\frac{1}{\epsilon}))$ time (the log factors are from the root-finding algorithm, e.g. the bisection method, for which the running time is logarithmic in the size of the interval being searched). This yields a total running time of $O(|\mathcal{S}|^2|\mathcal{A}|H + |\mathcal{S}||\mathcal{A}|(\log|\mathcal{A}| + \log|\mathcal{X}| + \log\frac{1}{\epsilon})$ for each iteration of Algorithm 6.1. In practice, we have observed that only a handful of iterations are required for convergence. By comparison, determining the optimal policy takes $O(|\mathcal{S}|^2|\mathcal{A}|H)$ time.

6.4.2 Analysis

In this section, we provide a proof that the sequence of estimates produced by Algorithm 6.1 converges to a limit that is a stationary point of $L(\boldsymbol{\pi})$, the function in Equation (6.1). This guarantee is similar to the one typically cited for the EM algorithm [19]; in fact, the convergence theorem used in the proof below is the same tool used by Wu [146] in his analysis of EM.

Theorem 6.1. *Algorithm 6.1 converges to a stationary point of* $L(\boldsymbol{\pi})$.

Proof. Let Ω be the set of all policies. We will need to assume that each maximization in Algorithm 6.1 finds a point in the interior of Ω (a similar assumption is made in Wu's proof of the convergence of the EM algorithm [146]). We can view Algorithm 6.1 as defining H distinct point-to-set maps $\{M_\tau\}_{\tau=1}^H$ on Ω, each corresponding to an optimization over a different $\boldsymbol{\pi}^\tau$. In other words, $\tilde{\boldsymbol{\pi}} \in M_\tau(\boldsymbol{\pi})$ if $\tilde{\boldsymbol{\pi}}$ is a solution to the problem of maximizing $L(\boldsymbol{\pi})$ over just the variables in $\boldsymbol{\pi}^\tau$ (recall that $\boldsymbol{\pi} = \left(\boldsymbol{\pi}^1, \ldots, \boldsymbol{\pi}^H\right)$). Let $M^A = M_H \circ M_{H-1} \cdots \circ M_1$, i.e., M^A is the point-to-set map defined by one complete cycle of optimizations.

By Convergence Theorem A from Zangwill [148], Algorithm 6.1 converges to a stationary point of L if: (a) Ω is compact, (b) for all $\tilde{\boldsymbol{\pi}} \in M^A(\boldsymbol{\pi})$, $L(\tilde{\boldsymbol{\pi}}) \geq L(\boldsymbol{\pi})$, (c) whenever $\boldsymbol{\pi}$ is not a stationary point of L, then for all $\tilde{\boldsymbol{\pi}} \in M^A(\boldsymbol{\pi})$, we have $L(\tilde{\boldsymbol{\pi}}) > L(\boldsymbol{\pi})$, and (d) M^A is a closed map.

Conditions (a), (b) and (c) are fairly straightforward to establish. The last condition (d) is more difficult, but this can be proved by observing that L is continuous, and then applying Proposition 7 and Theorem 8 from Hogan [48]. □

6.4.3 Unknown Transition Function

So far, we have assumed that the transition probabilities $\boldsymbol{\theta}$ of the modeling MDP are given. Removing this assumption presents no special difficulty, since it is possible for our algorithm to jointly estimate $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ within the framework already presented. The idea will be to define new state and action spaces $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{A}}$, and a new set of transition probabilities $\tilde{\boldsymbol{\theta}}$, in such a way that each parameter in the new set of unknowns $\tilde{\boldsymbol{\pi}}$ corresponds either to a parameter in $\boldsymbol{\pi}$ or a parameter in $\boldsymbol{\theta}$. Essentially, we fold the transition probabilities into the policy, and then replace them with a set of "dummy" transition probabilities. This reduction allows us to assume without loss of generality in our algorithm that $\boldsymbol{\theta}$ is known, and that everything unknown about the MDP is embodied in the policy $\boldsymbol{\pi}$.

Concretely, let $\tilde{\mathcal{S}} = \mathcal{S} \cup (\mathcal{S} \times \mathcal{A})$ and $\tilde{\mathcal{A}} = \mathcal{A} \cup \mathcal{S}$. We define $\tilde{\boldsymbol{\theta}}$ as

$$\tilde{\theta}^t_{\tilde{s}\tilde{a}\tilde{s}'} = \begin{cases} 1 & \text{if } \tilde{s} \in \mathcal{S},\ \tilde{a} \in \mathcal{A},\ \text{and } \tilde{s}' = (\tilde{s}, \tilde{a});\ \text{or} \\ & \text{if } \tilde{s} \in (\mathcal{S} \times \mathcal{A}),\ \tilde{a} \in \mathcal{S},\ \text{and } \tilde{s}' = \tilde{a} \\ 0 & \text{otherwise.} \end{cases}$$

Put differently, when we are in state $\tilde{s} = s$ and take action $\tilde{a} = a$, the environment deterministically transitions to "state" $\tilde{s}' = (s, a)$. And when we are in "state" $\tilde{s} = (s, a)$ and take "action" $\tilde{a} = s'$, the environment deterministically transitions to state $\tilde{s}' = s'$.

One last modification is needed: we define a new state $\tilde{s}^*$, with $R(\tilde{s}^*) = -\infty$, and set $\tilde{\theta}^t_{\tilde{s}\tilde{a}\tilde{s}^*} = 1$ whenever $\tilde{s}$ and $\tilde{a}$ do not make sense together, i.e., when $\tilde{s} \in \mathcal{S}$ and

$\tilde{a} \in \mathcal{S}$, or when $\tilde{s} \in (\mathcal{S} \times \mathcal{A})$ and $\tilde{a} \in \mathcal{A}$. This will force $\tilde{\pi}^t_{\tilde{s}\tilde{a}} = 0$ in these cases.

So we have the following equivalences between the old and new parameters:

$$\begin{aligned} \tilde{\pi}^t_{\tilde{s}\tilde{a}} &\Leftrightarrow \pi^t_{sa} && \text{if } \tilde{s} = s \text{ and } \tilde{a} = a \\ \tilde{\pi}^t_{\tilde{s}\tilde{a}} &\Leftrightarrow \theta^t_{sas'} && \text{if } \tilde{s} = (s, a) \text{ and } \tilde{a} = s' \end{aligned}$$

Note that, when applying this reduction, the prior $P(\tilde{\boldsymbol{\pi}}) = P(\boldsymbol{\pi}, \boldsymbol{\theta})$ assigns greater weight to policies and transition probabilities that *jointly* have high value.

6.5 Synthetic Experiments

Using synthetic environments, we compared the value-based prior to two existing algorithms for imitation learning. We also investigated our algorithm's sensitivity to the value of the mentor's policy. We review the other methods below, the synthetic environments in Section 6.5.1, and our experiments in Sections 6.5.2 and 6.5.3.

A number of authors have suggested methods to incorporate prior knowledge of the mentor's behavior into imitation learning. Price and Boutilier [99] described an approach based on the Dirichlet distribution. In their scheme, the policy at each state is assigned a prior distribution $P_s(a; \boldsymbol{\beta}) = \text{Dir}(\boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is a $|\mathcal{A}|$-length vector of positive reals. Let $\mathcal{A}^o_s$ be the set of optimal actions at state s. We define each $P_s(a; \boldsymbol{\beta})$ so that $\beta_a = \dfrac{\kappa}{|\mathcal{A}^o_s|}$. This amounts to asserting a prior belief that the mentor's policy is an optimal policy. Note that κ plays a similar role here as it does in Equation (6.1), in that it reflects the degree to which the prior is concentrated on high-value policies.

Henderson et al. [47] developed a modified temporal difference learning algorithm in which the usual Q values are adjusted so that the resulting optimal policy is forced to more closely match the mentor's behavior. Although it is difficult to describe succinctly, their algorithm employs a tunable parameter κ, which controls the trade-off between optimality and imitation, just as it does in our algorithm. Since TD

techniques do not assume that transition probabilities are given, we use the reduction described in Section 6.4.3 when comparing with our method.

6.5.1 Maze Environments

We used maze environments for all of our synthetic experiments. Each maze was a 30-by-30 grid, with the start state in one corner and the goal state, containing a large positive reward, in the opposite corner. Movement in a maze was in the four compass directions, but taking a move action risked a 30% chance of landing in a random adjacent cell. Also, obstacles (negative rewards) were randomly placed in 15% of the cells in each maze, with each having a magnitude that was, on average, 2/3 as large as the goal state's positive reward. Finally, the time horizon was set to 90, which was sufficient to allow even meandering policies to eventually reach the goal state.

Our environments had one additional feature that was introduced to make the comparison between the various algorithms more interesting. We found that the optimal action in each state typically had substantially larger value than any other action. So a prior that assigned greatest weight to the highest value policies essentially assigned greatest weight to a *single* policy, i.e., the policy that takes the optimal action in every state. In such circumstances, we did not expect to observe an advantage to using a value-based prior over a Dirichlet prior. To simulate a scenario where there are many diverse high-value policies, we introduced a "twin" action for every original action, i.e., a separate action that has exactly the same effect on the environment.

6.5.2 Comparison to Other Methods

For each maze environment, we generated data sets of state-action trajectories from an optimal policy for the maze.[2] However, when estimating that policy from data,

[2]Since there were always at least two optimal actions in each state, per Section 6.5.1, we randomly chose one of them to always take.

we supplied each algorithm with just the location and size of the goal reward, and *not* the locations or sizes of the obstacles. Effectively, each algorithm assigned the highest prior probability to a policy that moved directly towards the goal, ignoring obstacles altogether. So, from the perspective of each algorithm, the mentor's policy had high value, but was suboptimal.

Figure 6.1 compares the value-based prior to the Dirichlet prior suggested by Price and Boutilier [99]. First, note that our algorithm is much more robust to the value of the trade-off parameter κ; we varied κ over three orders of magnitude, and the value-based prior improved the accuracy of estimated policy throughout that range. This is important, as we are not proposing a principled way to set the value of κ, except to point out that it should generally increase with the value of the mentor's policy. Second, although the Dirichlet prior provided a more accurate estimate for smaller data sets for certain values of κ, that advantage soon became a disadvantage as the amount of data was increased. To understand why, recall that in our maze environment, there are many diverse policies that each have high value. The value-based prior assigns the same weight to every policy that has the same value, even if the policies themselves are quite different. But a Dirichlet prior is forced to encode the belief that a *particular* policy is most probable. If this policy differs from the mentor's policy, then it will skew the estimation, even if both are high value polices.

Figure 6.2 compares the value-based prior to the hybrid reinforcement/supervised learning algorithm proposed by Henderson et al. [47]. For the value-based prior, the reduction described in Section 6.4.3 was applied, since the hybrid algorithm does not assume that the transition probabilities given. Note that the value-based prior initially provides an inferior estimate than the naive method that uses no prior; this is because the algorithm at that stage is using poor approximations of the transition probabilities to compute value function in the modeling MDP. Nevertheless, as the number of samples increases, the value-based prior eventually provides an advantage.

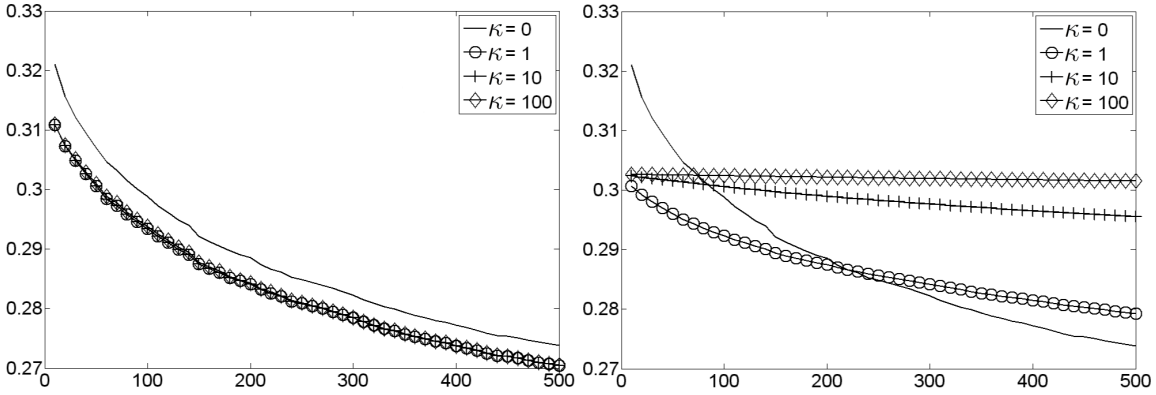

Figure 6.1: Left: Performance of the value-based prior. Right: Performance of the Dirichlet prior. The x-axis indicates the number of state-action trajectories in the data set, and the y-axis indicates the RMS error of the estimated policy with respect to the mentor's policy. Each line in each graph is the average estimation error for 50 mazes. κ is a trade-off parameter; $\kappa = 0$ corresponds to not using any prior at all.

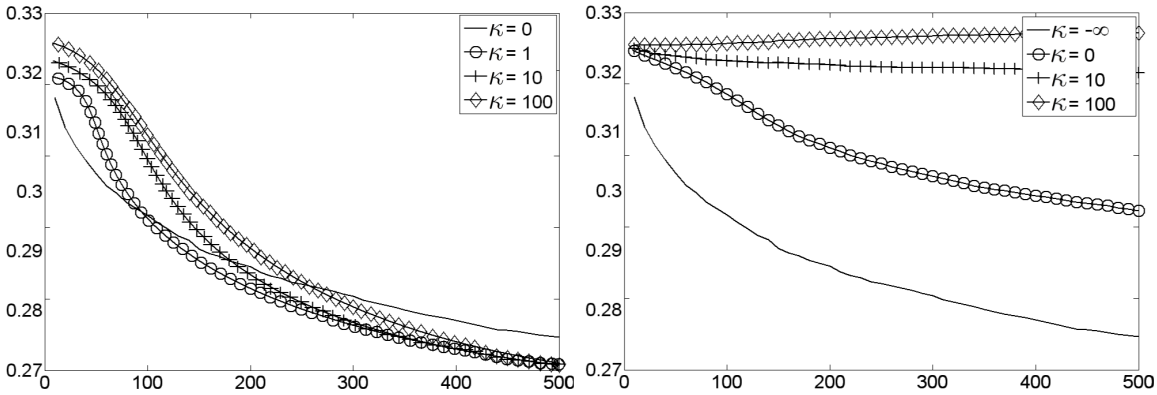

Figure 6.2: Left: Performance of the value-based prior. Right: Performance of the hybrid reinforcement/supervised learning algorithm. Details are the same as for Figure 6.1, except that for the value-based prior, the reduction described in Section 6.4.3 has been applied, and in the case of the hybrid algorithm, $\kappa = -\infty$ corresponds to ignoring rewards and simply imitating the behavior in the data.

6.5.3 Sensitivity to Policy Value

We also investigated how sensitive our algorithm is to the value of the mentor's policy.

To create policies with a variety of values, we used the following procedure. In each maze environment, we computed an optimal policy $\boldsymbol{\pi}^*$. We then randomly selected δ fraction of the states, and in each state swapped the optimal action in $\boldsymbol{\pi}^*$ with a randomly chosen action. We also added a small Gaussian perturbation (mean

0.5, variance σ^2) to each state-action probability, and renormalized appropriately. By carefully varying δ and σ^2, we were able to produce policies whose values were distributed in a range of 70% to 100% of the optimal value.

Figure 6.3 depicts the performance of our method for estimating policies with various values. As one might expect, performance degraded as the mentor's policy's value decreased. Nonetheless, we found that the value-based prior improves estimation even when the mentor's policy's value is reasonably far from optimal — as low as 80% of the optimal value.

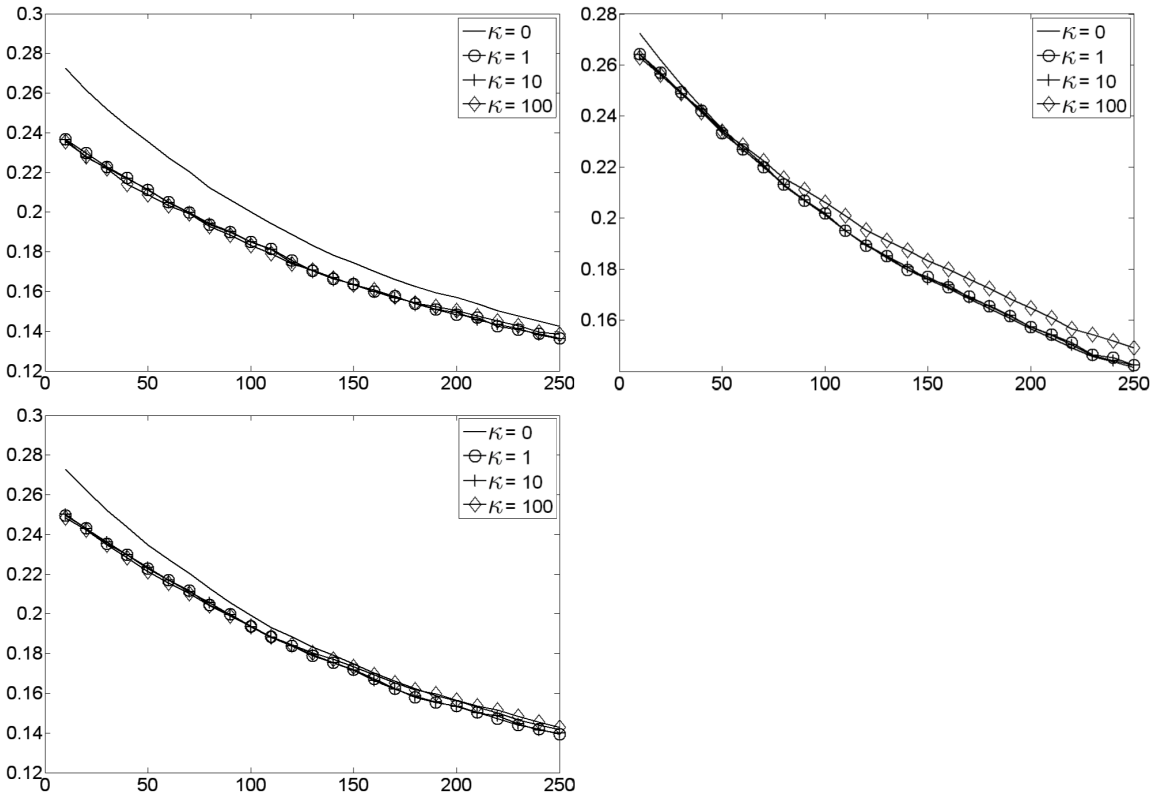

Figure 6.3: Performance of the value-based prior for policies with values approximately 70-90% of the optimal value. Axes and legend are the same as for Figure 6.1. Each line in each graph is the average estimation error for 50 policies (10 policies each from 5 maze environments). Top Left: Policies that have average value 89.6% of the optimal value, with std dev 1.3%. Bottom Left: Policies that have average value 80.9% of the optimal value, with std dev 3.2%. Top Right: Policies that have average value 72.9% of the optimal value, with std dev 1.1%.

6.6 Application to Dialog Modeling

When designing a dialog manager for a spoken dialog system, one would ideally like to try different dialog management strategies on the actual user population that will be using the system, and select the one that works best. However, users are typically unwilling to endure this kind of experimentation. Another approach is to build a model of user behavior, so that the designer can experiment exclusively with the model, without having to interact with (and potentially annoy) real users. Learning an accurate *user model* is the goal of this section.

When learning a user model, the value-based prior presented in Sections 6.2-6.4 can be used to encode the belief that users prefer to finish their dialogs as soon as possible. The main obstacle to directly applying such a prior to learning user models is that we typically do not observe human-computer dialogs themselves, but instead view noisy and incomplete versions of them. As an illustration, here is a short fragment of a dialog transcript from a voice-controlled telephone directory that is the subject of our experiments in Section 6.6.5:

$\tilde{s}_1 =$ `First and last name?`

$\tilde{a}_1 =$ *Jane Roe*

$\tilde{s}_2 =$ `Jane Roe.  Office or cell?`

$\tilde{a}_2 =$ *No, no, John Doe.*

$\tilde{s}_3 =$ `First and last name?`

...

Here $\tilde{s}_t$ is the system prompt at time t, and $\tilde{a}_t$ is the observed response by the user. This notation has been deliberately chosen to coincide with the notation for "state" and "action" in MDPs. However, $\tilde{s}_t$ is not the true state of the dialog, since the behavior of the user depends on more than just the last system prompt. Nor is

$\tilde{a}_t$ the true action, since the user's utterances will typically be corrupted by errors made by the automatic speech recognition (ASR) engine. Below is complete version of this dialog transcript, in which each observed $\tilde{s}_t$ and $\tilde{a}_t$ has been annotated with their true values s_t and a_t:

$\tilde{s}_1 =$ `First and last name?`

$s_1 =$ `First and last name?; Misunderstood = False`

$a_1 =$ "John Doe"

$\tilde{a}_1 =$ *Jane Roe*

$\tilde{s}_2 =$ `Jane Roe.  Office or cell?`

$s_2 =$ `Jane Roe.  Office or cell?; Misunderstood = True`

$a_2 =$ "No, no, John Doe"

$\tilde{a}_2 =$ *No*

$\tilde{s}_3 =$ `First and last name?`

...

Thus the state-action sequence $(\tilde{s}_1, \tilde{a}_1, \tilde{s}_2, \tilde{a}_2, \ldots)$ is a noisy version of the state-action sequence $(s_1, a_1, s_2, a_2, \ldots)$. In Section 6.6.1 we present a probabilistic graphical model for generating noisy dialog transcripts, and in Section 6.6.2 we explain how the well-known EM algorithm [19] can be used to estimate the unknown parameters of this model. These parameters specify a model of user behavior. In Section 6.6.3 we show how this approach can be extended to use a value-based prior, by using a version of EM called the ECM algorithm [78].

6.6.1 Graphical Model

We adopt a probabilistic graphical model of dialogs (similar to Williams and Young [145]), depicted schematically in Figure 6.4. Following the convention for graphical

models, we use directed edges to denote conditional dependencies among the variables. In our dialog model, a dialog transcript $\mathbf{x}$ consists of an alternating sequence of observed dialog states and observed user actions: $\mathbf{x} = (\tilde{s}_0, \tilde{a}_0, \tilde{s}_1, \tilde{a}_1, \ldots)$.

A dialog transcript $\mathbf{x}$ is generated by our model as follows: At each time t, the observed state is $\tilde{s}_t$ and the true state is s_t. The true state includes information about the user's hidden goal and relevant dialog history which, due to ASR confusions, is known with certainty only to the user. Conditioned on s_t, the user draws an unobserved action a_t from a distribution $\Pr(a_t \mid s_t; \boldsymbol{\pi})$ parameterized by an unknown parameter $\boldsymbol{\pi}$. This distribution is the user model.

For each user action a_t, the ASR engine produces a hypothesis $\tilde{a}_t$ of what the user said, drawn from a distribution $\Pr(\tilde{a}_t \mid a_t)$, called the ASR confusion model. The true state s_t is updated to s_{t+1} according to a distribution $\Pr(s_{t+1} \mid \tilde{s}_{t+1}, s_t, a_t, \tilde{a}_t)$. Then the next observed state $\tilde{s}_{t+1}$ is selected according to the dialog management policy.

Concretely, the values of $\tilde{s}_t, s_t, a_t$ and $\tilde{a}_t$ are all assumed to belong to finite sets, and so all the conditional distributions in our model are multinomials. Hence $\boldsymbol{\pi}^t$ is a vector that parameterizes the user model according to $\Pr(a_t = a \mid s_t = s; \boldsymbol{\pi}) = \pi^t_{sa}$.

The problem we are interested in is estimating $\boldsymbol{\pi}$ given the set of dialog transcripts $\mathcal{X}$, $\Pr(\tilde{a}_t \mid a_t)$ and $\Pr(s_{t+1} \mid \tilde{s}_{t+1}, s_t, a_t, \tilde{a}_t)$. Here, we assume that $\Pr(\tilde{a}_t \mid a_t)$ is relatively straightforward to estimate: For example, ASR models that rely a simple confusion rate and uniform substitutions (which can be estimated from small number of transcriptions) have been used to train dialog systems which outperform traditional systems [131]. Further, $\Pr(s_{t+1} \mid \tilde{s}_{t+1}, s_t, a_t, \tilde{a}_t)$ is often deterministic and tracks dialog history relevant to action selection — for example, whether the system correctly or incorrectly confirms a slot value. Here we assume that it can be easily hand-crafted. Notice that $\boldsymbol{\pi}$ has the form of a policy in a finite-horizon MDP.

We consider two definitions of a policy $\hat{\boldsymbol{\pi}}$ that best models user behavior. One is

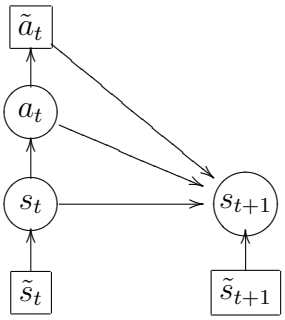

Figure 6.4: A probabilistic graphical model of a human-computer dialog. The boxed variables are observed; the circled variables are unobserved.

the maximum likelihood estimate:

$$\hat{\boldsymbol{\pi}} = \arg\max_{\boldsymbol{\pi}} \log \Pr(\mathcal{X} \mid \boldsymbol{\pi}) \tag{6.5}$$

and the other is the MAP estimate with respect to a value-based prior:

$$\hat{\boldsymbol{\pi}} = \arg\max_{\boldsymbol{\pi}} \log \Pr(\mathcal{X} \mid \boldsymbol{\pi}) + \kappa V(\boldsymbol{\pi}) \tag{6.6}$$

Computing either estimate is complicated by the fact that $\mathcal{X}$ contains only the observed states and actions, and not the true states and actions. The next two sections describe algorithms for computing (6.5) and (6.6), respectively.

6.6.2 EM Algorithm

In general, computing the maximum likelihood estimate $\hat{\boldsymbol{\pi}}$ in (6.5) exactly is intractable. However, we can efficiently approximate $\hat{\boldsymbol{\pi}}$ via an expectation-maximization (EM) procedure [19]. The EM algorithm is really a meta-algorithm that must be tailored to the particular probabilistic model to which it is applied. In this section, we derive the EM algorithm for the model in Figure 6.4.

For a dialog transcript $\mathbf{x}$, let $\mathbf{y}$ be the corresponding sequence of unobserved values: $\mathbf{y} = (s_1, a_1, s_2, a_2, \ldots)$. Let $\mathcal{Y}$ be the set of all sequences of unobserved values

corresponding to the data set $\mathcal{X}$.

In each iteration i of the EM algorithm, we compute the expectation of the *unobserved* data log-likelihood with respect to the estimate of $\boldsymbol{\pi}^{i-1}$ from the previous iteration (called the *E-step*). Then we maximize this expectation over $\boldsymbol{\pi}$ to obtain the new estimate $\boldsymbol{\pi}^i$ (the *M-step*). More concretely

$$\begin{aligned} \boldsymbol{\pi}^i &= \arg\max_{\boldsymbol{\pi}} \; E\left[\log \Pr(\mathcal{Y} \mid \boldsymbol{\pi}) \;\middle|\; \mathcal{X}, \boldsymbol{\pi}^{i-1}\right] \\ &\triangleq \arg\max_{\boldsymbol{\pi}} Q(\boldsymbol{\pi}, \boldsymbol{\pi}^{i-1}) \end{aligned}$$

Recall that $\Pr(\mathcal{X} \mid \boldsymbol{\pi})$ is the likelihood of the observed data. It can be shown that the following are all true [146]:

1. $Q(\boldsymbol{\pi}, \boldsymbol{\pi}^{i-1}) \leq \Pr(\mathcal{X} \mid \boldsymbol{\pi})$ for all $\boldsymbol{\pi}$.
2. $Q(\boldsymbol{\pi}^{i-1}, \boldsymbol{\pi}^{i-1}) = \Pr(\mathcal{X} \mid \boldsymbol{\pi}^{i-1})$.
3. $\left.\dfrac{\partial Q(\boldsymbol{\pi}, \boldsymbol{\pi}^{i-1})}{\partial \boldsymbol{\pi}}\right|_{\boldsymbol{\pi}=\boldsymbol{\pi}^{i-1}} = \left.\dfrac{\partial \log \Pr(\mathcal{X} \mid \boldsymbol{\pi})}{\partial \boldsymbol{\pi}}\right|_{\boldsymbol{\pi}=\boldsymbol{\pi}^{i-1}}$

These properties together imply that the EM algorithm monotonically increases $\Pr(\mathcal{X} \mid \boldsymbol{\pi})$, and that if the algorithm ever converges, then it converges to the set of stationary points of $\Pr(\mathcal{X} \mid \boldsymbol{\pi})$.

It remains to argue that the algorithm actually does converge; Wu [146] described a set of fairly mild conditions that ensure that it does.

Of course, EM is not guaranteed to converge to the global maximum of the likelihood. However, it usually converges to a local maximum (though in certain pathological cases, it can converge to a saddle point, or even a local minimum).

E-Step and M-Step Derivation

Recall that the E-step of the EM algorithm involves computing the expression for $Q(\boldsymbol{\pi}, \boldsymbol{\pi}^{i-1})$, while the M-step is to maximize $Q(\boldsymbol{\pi}, \boldsymbol{\pi}^{i-1})$ with respect to $\boldsymbol{\pi}$. For our

model, the E-step reduces to being able to compute, for any state s, action a, dialog $\mathbf{x}$, and parameters $\boldsymbol{\pi}$, the probability

$$\Pr(s_t = s, a_t = a \mid \mathbf{x}, \boldsymbol{\pi})$$

We will do this efficiently by using the *forward-backward* algorithm, originally derived for Hidden Markov Models [102]. If we define a forward variable $F_t(s, a)$ and a backward variable $B_t(s, a)$ as

$$F_t(s, a) \triangleq \Pr(\mathbf{x}_{1:t}, s_t = s, a_t = a \mid \boldsymbol{\pi})$$
$$B_t(s, a) \triangleq \Pr(\mathbf{x}_{t+1:T} \mid s_t = s, a_t = a, \boldsymbol{\pi})$$

Then we have

$$\Pr(s_t = s, a_t = a \mid \mathbf{x}, \boldsymbol{\pi}) = \frac{F_t(s, a) B_t(s, a)}{\sum_{s', a'} F_t(s', a') B_t(s', a')}$$

It can be shown that the forward variable can be computed inductively

$$F_1(s, a) = \alpha_s \cdot \pi^1_{sa} \cdot \Pr(\tilde{a}_1 \mid a)$$
$$F_t(s, a) = \sum_{s', a'} F_{t-1}(s', a') \cdot \theta_{s'a's} \cdot \pi^t_{sa} \cdot \Pr(\tilde{a}_1 \mid a)$$

and so can the backward variable

$$B_T(s, a) = 1$$
$$B_t(s, a) = \sum_{s', a'} \theta_{sas'} \cdot \pi^{t+1}_{s'a'} \cdot \Pr(\tilde{a}_{t+1} \mid a') \cdot B_{t+1}(s', a')$$

Now we can give an expression for $Q(\boldsymbol{\pi}, \boldsymbol{\pi}^{i-1})$. Let

$$C_{sat}^{i-1} \triangleq \sum_d \Pr(s_t^d = s, a_t^d = a \mid \mathbf{x}^d, \boldsymbol{\pi}^{i-1})$$

where $\mathbf{x}^d$ is the dth dialog in the set $\mathcal{X}$, and (s_t^d, a_t^d) is the state-action pair in the tth time step of that dialog. Then we have

$$Q(\boldsymbol{\pi}, \boldsymbol{\pi}^{i-1}) = \sum_{s,a,t} C_{sat}^{i-1} \log \pi_{sa}^t$$

and this is the quantity we want to maximize over $\boldsymbol{\pi}$ in the M-step of the EM algorithm, subject to the stochastic constraints $\sum_a \pi_{sa}^t = 1$ and $\pi_{sa}^t \geq 0$. Therefore the solution to the M-step is given by the simple computation

$$\pi_{sa}^t = \frac{C_{sat}^{i-1}}{\sum_{a'} C_{sa't}^{i-1}}$$

6.6.3 ECM Algorithm

We use a variant of the EM algorithm to compute the MAP estimate $\hat{\pi}$ in (6.6). If we repeat the derivation of the E-step and M-step from the previous section for the objective in (6.6), we find that the E-step is unchanged, and the M-step is to maximize

$$Q(\boldsymbol{\pi}, \boldsymbol{\pi}^{i-1}) = \sum_{s,a,t} C_{sat}^{i-1} \log \pi_{sa}^t + \kappa V(\boldsymbol{\pi}) \tag{6.7}$$

over $\boldsymbol{\pi}$. Notice that here Q is identical to the objective from Section 6.2. That is, the policy that maximizes Q is the MAP estimate with respect to the value-based prior, except that the observations K_{sat} have been replaced by the "expected" observations C_{sat}^{i-1}. Recall that Algorithm 6.1, the alternating maximization algorithm, was designed to find this estimate.

So, at least at a high-level, we can use Algorithm 6.1 for the M-step. However,

there is a technical complications with this approach: Recall that Algorithm 6.1 is only guaranteed to converge *asymptotically* to a *stationary point* of the objective in (6.7). This is a substantially weaker than *maximizing* the objective, which is what the analysis of the EM algorithm requires.

One solution to this complication is to simply ignore it. Since Algorithm 6.1 is an alternating maximization, it might be reasonable to execute one cycle of these maximizations (i.e., a total of H iterations) in each M-step, and hope for convergence.

In fact, it is possible to prove that the approach described above is theoretically sound. We will do so by directly applying the analysis of the ECM algorithm [78], a variant of the EM algorithm designed for problems in which the M-step of the EM algorithm involves a difficult maximization. The idea of the ECM algorithm is to replace the difficult maximization with a sequence of simpler maximizations. The most basic version of the ECM approach partitions the variables in the problem into several subsets, and then maximizes over each subset, in turn, while holding the remaining variables fixed (these simpler maximizations are called "conditional maximizations", hence the name "ECM", though it might be more fitting to call them "constrained maximizations"). It was shown by Meng and Rubin [78] that the ECM algorithm converges to the set of stationary points of the likelihood, under essentially the same conditions that the EM algorithm does.

We can apply the ECM algorithm to our problem by partitioning $\boldsymbol{\pi}$ into H subsets

$$\boldsymbol{\pi}^t = \{\pi^t_{sa} \mid s \in \mathcal{S}, a \in \mathcal{A}\},$$

for each $t = 1, \ldots, H$. In other words, the M step is replaced by H conditional maximization steps (CM-steps), and in the t^{th} CM-step, we will maximize over just the policy at time t. Here is an outline of one E-step and the corresponding cycle of CM-steps of the ECM algorithm:

$$
\begin{aligned}
\text{E-step:} \quad & \text{Compute constants } C_{sat} \text{ using the} \\
& \text{forward-backward algorithm.} \\
\text{CM-step } (H): \quad & \max Q(\boldsymbol{\pi}^H) + \kappa V(\boldsymbol{\pi}^H) \\
\text{CM-step } (H-1): \quad & \max Q(\boldsymbol{\pi}^{H-1}) + \kappa V(\boldsymbol{\pi}^{H-1}) \\
& \vdots \\
\text{CM-step } (1): \quad & \max Q(\boldsymbol{\pi}^1) + \kappa V(\boldsymbol{\pi}^1)
\end{aligned}
$$

Here we are using the arguments to Q and V to denote what the free variables are in each maximization. In each CM-step, the maximizing values from the previous CM-step are used as the values for the fixed variables. Importantly, observe that this algorithm is identical to the procedure that we informally justified above.

6.6.4 Target Application

We will apply the algorithms described in Sections 6.6.2-6.6.3 to a voice-controlled telephone directory. This system is currently in use in a large company with many thousands of employees. Users call the directory system and provide the name of a callee they wish to be connected to. The system then requests additional information from the user, such as the callee's location and type of phone (office or cell). A fragment from a dialog with the system was given at the beginning of Section 6.6. Because the telephone directory has many names, the number of possible values for user actions a_t and dialog states s_t is potentially very large. To control the size of the model, we first assumed that the user's intended callee does not change during the call, which allows us to group many user actions together into generic placeholders e.g. a_t = `FirstNameLastName`. After doing this, there were a total of 13 possible values for a_t and 112 values for s_t. We also fixed the maximum length of any dialog (i.e., the horizon length) to 10 turns.

In addition to the system prompt, the dialog state consists of three bits: one bit indicating whether the system has correctly recognized the callee's name, one bit indicating whether the system has correctly recognized the callee's "phone type" (office or cell), and one bit indicating whether the user has said the callee's geographic location (needed for disambiguation when several different people share the same name). The deterministic distribution $\Pr(s_{t+1} \mid \tilde{s}_{t+1}, s_t, a_t, \tilde{a}_t)$ simply updates the user state after each dialog turn in the obvious way. For example, the "name is correct" bit of s_{t+1} is set to 0 whenever the system prompt is a confirmation of a name which doesn't match a_t.

Recall that the user model is a multinomial distribution $\Pr(a_t \mid s_t; \boldsymbol{\pi})$ parameterized by a vector $\boldsymbol{\pi}$. Based on the number user actions, dialog states, and the maximum length of a dialog, $\boldsymbol{\pi}$ is a vector of 1344 unknown parameters for our target application.

6.6.5 Experiments

We conducted two sets of experiments on the telephone directory application, one using simulated data, and the other using dialogs collected from actual users. Both sets of experiments assumed that all the distributions in Figure 6.4, except the user model, are known. The ASR confusion model was estimated by transcribing 50 randomly chosen dialogs from the training set in Section 6.6.5 and calculating the frequency with which the ASR engine recognized $\tilde{a}_t$ such that $\tilde{a}_t \neq a_t$. The probabilities $\Pr(\tilde{a}_t \mid a_t)$ were then constructed by assuming that, when the ASR engine makes an error recognizing a user action, it substitutes another randomly chosen action.

Simulated Data

Recall that, in our parameterization, the user model is $\Pr(a_t = a \mid s_t = s; \boldsymbol{\pi}) = \pi^t_{sa}$. So in this set of experiments, we chose a reasonable, hand-crafted value for $\boldsymbol{\pi}$, and

then generated synthetic dialogs by following the probabilistic process depicted in Figure 6.4. In this way, we were able to create synthetic training sets of varying sizes, as well as a test set of 1000 dialogs. Each generated dialog $\mathbf{d} = (\mathbf{x}, \mathbf{y})$ in each training/test set consisted of the values for all the observed *and* unobserved variables.

For a training/test set $\mathcal{D}$, let $K^{\mathcal{D}}_{sat}$ be the number of times, in all the dialogs in $\mathcal{D}$, that $a_t = a$ and $s_t = s$. Similarly, let $\widetilde{K}^{\mathcal{D}}_{sat}$ be the number of times that $\tilde{a}_t = a$ and $\tilde{s}_t = s$.

For each training set $\mathcal{D}$, we estimated $\boldsymbol{\pi}$ using the following three methods:

1. *Manual*: Let $\boldsymbol{\pi}$ be the maximum likelihood estimate using manually transcribed dialogs, i.e., $\pi^t_{sa} = \frac{K^{\mathcal{D}}_{sat}}{\sum_{a'} K^{\mathcal{D}}_{sa't}}$.

2. *Automatic*: Let θ be the maximum likelihood estimate using dialogs that were automatically transcribed by the ASR engine, i.e., $\pi^t_{sa} = \frac{\widetilde{K}^{\mathcal{D}}_{sat}}{\sum_{a'} \widetilde{K}^{\mathcal{D}}_{sa't}}$. This approach ignores ASR transcription errors, and also assumes that user behavior depends only on the observed data.

3. *EM*: Let $\boldsymbol{\pi}$ be the estimate produced by the EM algorithm described in Section 6.6.2.

Now let $\mathcal{D}'$ be the test set. We evaluated each user model by calculating the normalized log-likelihood of the model with respect to the *true* user actions in $\mathcal{D}'$:

$$\ell(\boldsymbol{\pi}) = \frac{\sum_{s,a,t} K^{\mathcal{D}'}_{sat} \log \pi^t_{sat}}{|\mathcal{D}'|}$$

$\ell(\boldsymbol{\pi})$ is essentially a measure of how well the user model parameterized by $\boldsymbol{\pi}$ replicates the distribution of user actions in the test set. The normalization is to allow for easier comparison across data sets of differing sizes.

We repeated this entire process (generating training and test sets, estimating and evaluating user models) 50 times. The results presented in Figure 6.5 are the average

of those 50 runs. They are also compared to the normalized log-likelihood of the "Truth", which is the actual parameter $\boldsymbol{\pi}$ used to generated the data.

The EM method has to estimate a larger number of parameters than the Automatic method (1344 vs. 168). But as Figure 6.5 shows, after observing enough dialogs, the EM method is able to leverage the hidden user state to learn a better model of user behavior, with an average normalized log-likelihood that falls about halfway between that of the models produced by the Automatic and Manual methods.

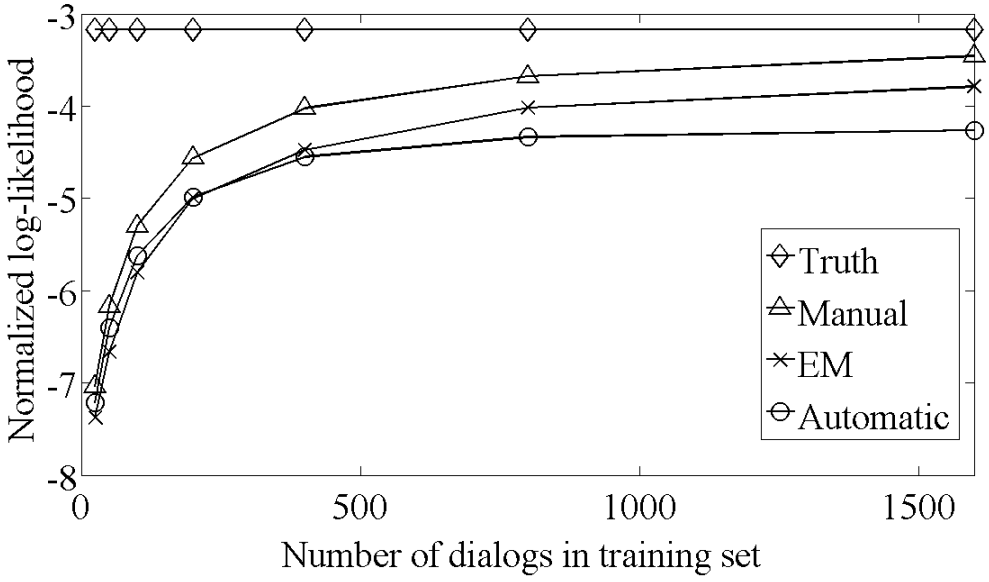

Figure 6.5: **Normalized log-likelihood of each model type with respect to the test set vs. size of training set.** Each data point is the average of 50 runs. For the largest training set, the EM models had higher normalized log-likelihood than the Automatic models in 48 out of 50 runs.

Real Data

We tested the three estimation methods from the previous section on a data set of 461 real dialogs, which we split into a training set of 315 dialogs and a test set of 146 dialogs. All the dialogs were both manually and automatically transcribed, so that each of the three methods was applicable. We also tested a fourth method:

4. *EM+Prior*: Let $\boldsymbol{\pi}$ be the estimate produced by the ECM algorithm described

in Section 6.6.3. For the value-based prior, we used a simple reward function that penalizes the length of a dialog, thereby encoding our belief that users generally prefer shorter dialogs.

The normalized log-likelihood of each user model, with respect to both the training and test set, is given in Table 6.1. Since the output of the EM and EM+Prior methods depend on a random choice of starting point $\boldsymbol{\pi}^{(0)}$, those results were averaged over 50 runs. Although the EM+Prior method (unsurprisingly) achieved slightly lower likelihood on the training set than the EM method, it achieved higher likelihood on the test set, indicating that the value-based prior helped bias the estimation towards a better model.

	Training Set $\ell(\theta)$	Test Set $\ell(\theta)$
Manual	-2.87	-3.73
EM	-3.90	-4.33
EM+Prior	-3.95	-4.15
Automatic	-4.60	-5.80

Table 6.1: **Normalized log-likelihood of each model type with respect to the training set and the test set.** The EM values are the average of 50 runs. Both the EM and EM+Prior models had higher normalized log-likelihood than the Automatic model in 50 out of 50 runs.

6.7 Other Related Work

Imitation learning has been studied extensively in robotics, where it is usually called *learning from demonstration*; an excellent survey is given by Argall et al. [3].

In Section 6.5, we reviewed a Bayesian approach to imitation learning due to Price and Boutilier [99]; earlier work by the same authors used observations from a mentor to supplement exploration in reinforcement learning [97], including scenarios where the mentor and imitator do not share the same set of actions [98].

The hybrid imitation/reinforcement learning algorithm of Henderson et al. [47], which we described in Section 6.5, is essentially a modified Q-learning algorithm.

Recently, Fern et al. [28] proposed a similar yet simpler method that uses a Boltzmann distribution to assign greater prior probability to mentor actions that have higher Q values.

In Section 6.6.1 we described a probabilistic graphical model for imitating users in a spoken dialog system; Shon et al. [119] and Verma and Rao [136] also studied using graphical models for imitation learning.

The idea of learning a user model in a spoken dialog system from transcribed dialogs has been studied by many authors [36, 69, 94, 115]. However, prior to the work described in this chapter, only Schatzmann et al. [115] had applied the EM algorithm to this problem, and even then assumed that the transcriptions were error-free.

6.8 Conclusion

In this chapter, we described an approach to imitation learning in which the imitator uses the value function of an MDP to assert a prior belief on a mentor's policy. We proved the convergence of an efficient algorithm to a stationary point of an appropriately defined posterior distribution. Synthetic experiments indicated that a value-based prior is robust in at least two senses: it is effective over a wide range of values for the trade-off parameter, and it is effective even when the mentor's policy is suboptimal. Finally, experiments on a spoken dialog system showed that a value-based prior can accelerate the accurate imitation of a goal-directed mentor.

Chapter 7

Conclusion

We now briefly review the main contributions of this thesis, and also suggest avenues for future work.

In Chapter 2, we explained that any no-regret algorithm can be used to compute minimax/maximin strategies in a two-player zero-sum game, and also that any algorithm that suffers no internal (or swap) regret can be used to compute a correlated equilibrium in an n-player game. In Chapter 3, we were able to deepen these well-known connections between no-regret algorithms and game theory, by showing that the MW algorithm converges to a lexicographic maximin strategy in a two-player zero-sum game whenever the game satisfies a certain technical condition. Further, we described a set of properties that, if satisfied by *any* no-regret algorithm, guarantee convergence to a lexicographic maximin strategy.

In Chapter 4, we introduced the apprenticeship learning framework, and presented two new apprenticeship learning algorithms that are based on mimicking the mentor, which is the most common approach to apprenticeship learning. Most apprenticeship learning algorithms reduce the problem to reinforcement learning, and consequently they suffer from the same challenges of large state spaces, exploration vs. exploitation trade-offs, etc. This fact is somewhat contrary to the intuition that demonstrations

from a mentor — especially a good mentor — should make the problem easier, not harder. So we described a method of reducing apprenticeship learning to classification, which is a much better studied problem for which there exist algorithms with strong performance guarantees.

In Chapter 5, we showed that the apprenticeship learning problem can be naturally formulated as a two-player zero-sum game, and observed that the MW algorithm is especially appropriate for solving this game, because the time it requires to solve the game does not depend on the exponential number of strategies available to one of the players. One of the major advantages of our game-theoretic formulation was that it freed us from having to mimic the mentor, which we showed can result in a much better apprentice policy when the mentor is behaving poorly and the apprentice can exploit prior knowledge about the true rewards contained in the features. Furthermore, our algorithm and analysis leveraged our results from Chapter 3 about lexicographic optimality. In particular, the connection between the MW algorithm and lexicographic optimality allowed us to compute a much simpler apprentice policy in certain cases, provided a robustness to changes in the scales of the features, and explained some of our experimental observations.

In the preceding chapters, we described algorithms for maximizing reward when the reward function is only partially known. In Chapter 6, instead of maximizing reward, our goal was to use the reward function to bias the imitation of an observed mentor, by asserting an *a priori* belief that the mentor is reward-seeking. This is an unorthodox use of a reward function, and we saw that it is especially appropriate for learning a user model in a spoken dialog system, because users is that setting are aptly described as being reward-seeking.

There are many opportunities for extending the results described in this thesis, and we now highlight some of the more promising directions.

Our results about the convergence of no-regret algorithms to a lexicographic max-

imin strategy in a zero-sum game only apply in special cases — either when the game satisfies certain conditions, or the no-regret algorithm does. This is in marked contrast to existing results that prove that all no-regret algorithms converge to minimax/maximin strategies in any zero-sum game. So a natural opportunity for future work is to generalize these results, or to show that they cannot be generalized.

One of the drawbacks of many of our apprenticeship learning algorithms is that they output mixed policies, which have a complicated and unnatural structure. We described a few solutions to this problem, but the most general solution, based on linear programming, only applied when the state space is finite and the transition function is known. It is still an open problem whether there is an efficient algorithm that solves the game-theoretic formulation of the apprenticeship learning problem and outputs a *single* stationary policy, even when the state space is infinite (our reduction of apprenticeship learning to classification is a candidate for such an algorithm, except it does not apply to the game-theoretic formulation).

Most of our apprenticeship learning algorithms use a reinforcement learning algorithm as a subroutine, and the only difference between calls to the subroutine is a change in the reward function. Moreover, consecutive calls typically do not change the reward function very much. Further, most reinforcement learning algorithms, such as value iteration and policy iteration, perform better when they are initialized with a policy that is "close" (in some sense) to the optimal policy. This discussion is obviously suggestive, and one should be able to obtain better performance guarantees by exploiting these properties.

Finally, the apprenticeship learning framework is based on observing a single mentor. But in many cases, there may be a committee of mentors that an apprentice can observe. It is unclear what the goal of the apprentice should be in this scenario — should she aim to perform as well as the best mentor, the worst mentor, or the average mentor — nor is it obvious how the apprentice can most efficiently achieve any

of these goals.

Bibliography

[1] Pieter Abbeel and Andrew Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 1–8, 2004.

[2] David Andre and Astro Teller. Evolving team Darwin United. In *RoboCup-98: Robot Soccer World Cup II*, pages 346–351, 1998.

[3] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics Autonomous Systems*, 57(5):469–483, 2009.

[4] Christopher G. Atkeson and Stefan Schaal. Robot learning from demonstration. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 12–20, 1997.

[5] J. Andrew Bagnell, Sham Kakade, Andrew Y. Ng, and Jeff Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems 17*, 2003.

[6] Leon Barrett and Srini Narayanan. Learning all optimal policies with multiple criteria. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pages 41–47, 2008.

[7] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[8] Dimitri P. Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983.

[9] Navin A. R. Bhat and Kevin Leyton-Brown. Computing Nash equilibria of action-graph games. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 35–42, 2004.

[10] David Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, 1956.

[11] Doron Blatt and Alfred Hero. From weighted classification to policy search. In *Advances in Neural Information Processing Systems 20*, pages 139–146, 2006.

[12] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[13] Ronen I. Brafman and Moshe Tennenholtz. R-MAX — A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2001.

[14] Leo Breiman. Prediction games and arcing classifiers. *Neural Computation*, 11 (7):1493–1517, 1999.

[15] George Brown. Some notes on computation of games solutions. Technical Report RAND Report P-78, The RAND Corporation, 1949.

[16] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[17] George Dantzig. Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation*, pages 339–347, 1951.

[18] George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B*, 39(1):1–38, 1977.

[20] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[21] Dmitri Dolgov and Edmund Durfee. Stationary deterministic policies for constrained MDPs with multiple rewards, costs, and discount factors. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1326–1331, 2005.

[22] Melvin Dresher. *Games of strategy*. Prentice Hall, 1961.

[23] Eyal Even-Dar, Sham M. Kakade, and Yishay Mansour. Experts in a Markov decision process. In *Advances in Neural Information Processing Systems 19*, pages 401–408, 2005.

[24] Shu-Cherng Fang and Sarat Puthenpura. *Linear Optimization and Extensions: Theory and Algorithms*. Prentice Hall, 1993.

[25] Eugene A. Feinberg and Adam Schwartz. *Handbook of Markov Decision Processes: Methods and Applications*. Springer, 2002.

[26] Eugene A. Feinberg and Adam Shwartz. Constrained Markov decision models with weighted discounted rewards. *Mathematics of Operations Research*, 20(2): 302–320, 1995.

[27] Eugene A. Feinberg and Adam Shwartz. Constrained discounted dynamic programming. *Mathematics of Operations Research*, 21(4):922–945, 1996.

[28] Alan Fern, Sriraam Natarajan, Kshitij Judah, and Prasad Tadepalli. A decision-theoretic model of assistance. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 1879–1884, 2007.

[29] Dean Foster and Rakesh Vohra. Asymptotic calibration. *Biometrika*, 85(2): 379–390, 1996.

[30] Yoav Freund and Robert E. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 325–332, 1996.

[31] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[32] Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999.

[33] Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. Multi-criteria reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 197–205, 1998.

[34] Ashutosh Garg and Dan Roth. Margin distribution and learning algorithms. In *Proceedings of the Twentieth International Conference on Machine Learning*, volume 20, 2003.

[35] Ashutosh Garg, Sariel Har-Peled, and Dan Roth. On generalization bounds, projection profile, and margin distribution. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 171–178, 2002.

[36] Kallirroi Georgila, James Henderson, and Oliver Lemon. User simulation for

spoken dialogue systems: Learning and evaluation. In *Proceedings of the International Conference on Spoken Language Processing*, 2006.

[37] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem — Part II. *Operations Research*, 11(6):863–888, 1963.

[38] Geoff Gordon. *Approximate Solutions to Markov Decison Processes*. PhD thesis, Carnegie Mellon University, 1999.

[39] Geoffrey J. Gordon, Amy R. Greenwald, and Casey Marks. No-regret learning in convex games. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pages 360–367, 2008.

[40] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming (web page and software), 2008. http://stanford.edu/∼boyd/cvx.

[41] Peter J. Green. Penalized likelihood. In *Encyclopedia of Statistical Sciences, Update Volume 2*, pages 578–586. 1996.

[42] Amy Greenwald and Amir Jafari. A general class of no-regret learning algorithms and game-theoretic equilibria. pages 1–11, 2003.

[43] James Hannan. Approximation to Bayes risk in repeated play. In *Contributions to the Theory of Games, Volume III*, pages 97–139. 1957.

[44] Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[45] David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1): 78–150, 1992.

[46] Elad Hazan and Satyen Kale. Computational equivalence of fixed points and no regret algorithms, and convergence to equilibria. In *Advances in Neural Information Processing Systems 21*, pages 625–632, 2007.

[47] James Henderson, Oliver Lemon, and Kallirroi Georgila. Hybrid reinforcement/supervised learning for dialogue policies from Communicator data. In *Proceedings of the Workshop on Knowledge and Reasoning in Practical Dialogue Systems, International Joint Conference on Artificial Intelligence*, pages 68–75, 2005.

[48] William W. Hogan. Point-to-set maps in mathematical programming. *SIAM Review*, 15(3):591–603, 1973.

[49] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[50] Ronald A. Howard. *Dynamic Programming and Markov Process*. MIT Press, 1960.

[51] Michael Johanson, Martin Zinkevich, and Michael H. Bowling. Computing robust counter-strategies. In *Advances in Neural Information Processing Systems 21*, pages 721–728, 2007.

[52] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274, 2002.

[53] Shizuo Kakutani. A generalization of Brouwer's fixed point theorem. *Duke Mathematical Journal*, 8(3):416–427, 1941.

[54] Adam Kalai and Santosh Vempala. Geometric algorithms for online optimiza-

tion. Technical Report MIT-LCS-TR-861, Massachusettes Institute of Technology, 2002.

[55] Narendra Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[56] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

[57] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[58] Michael J. Kearns, Robert E. Schapire, and Linda M. Sellie. Toward efficient agnostic learning. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 341–352, 1992.

[59] Michael J. Kearns, Michael L. Littman, and Satinder P. Singh. Graphical models for game theory. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 253–260, 2001.

[60] Leonid Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244(5):1093–1096, 1979.

[61] Victor Klee and George Minty. How good is the simplex algorithm? In *Inequalities III*, pages 159–175. 1972.

[62] Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1-2):167–215, 1997.

[63] Daphne Koller, Nimrod Megiddo, and Bernhard Von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 750–759, 1994.

[64] J. Zico Kolter, Pieter Abbeel, and Andrew Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. In *Advances in Neural Information Processing Systems 22*, pages 769–776, 2008.

[65] Michail G. Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 424–431, 2003.

[66] John Langford and Bianca Zadrozny. Relating reinforcement learning performance to classification performance. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 473–480, 2005.

[67] Emanuel Laskar. *The Brooklyn Daily Eagle*, page 53, June 1902.

[68] C. E. Lemke and Jr. J. T. Howson. Equilibrium points of bimatrix games. *SIAM Journal on Applied Mathematics*, 12(2):413–423, 1964.

[69] Esther Levin, Roberto Pieraccini, and Wieland Eckert. A stochastic model of human-machine interaction for learning dialogue strategies. *IEEE Transactions on Speech and Audio Processing*, 8(1):11–23, 2000.

[70] Kevin Leyton-Brown and Moshe Tennenholtz. Local-effect games. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 772–780, 2003.

[71] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[72] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.

[73] Michael L. Littman and Csaba Szepesvri. A generalized reinforcement-learning model: Convergence and applications. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 310–318, 1996.

[74] Michael L. Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 16*, pages 1555–1561, 2002.

[75] Shie Mannor and Nahum Shimkin. A geometric approach to multi-criterion reinforcement learning. *Journal of Machine Learning Research*, 5:325–360, 2004.

[76] Peter McCracken and Michael Bowling. Safe strategies for agent modelling in games. In *AAAI Fall Symposium on Artificial Multi-agent Learning*, pages 103–110, 2004.

[77] H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 536–543, 2003.

[78] Xiao-Li Meng and Donald B. Rubin. Maximum likelihood estimation via the ECM algorithm: A general framework. *Biometrika*, 80(2):267–278, 1993.

[79] Peter Miltersen and Troels Sorensen. Computing proper equilibria of zero-sum games. In *Proceedings of the Fifth International Conference on Computers and Games*, pages 200–211, 2006.

[80] Peter Miltersen and Troels Sorensen. Fast algorithms for finding proper strategies in game trees. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 874–883, 2008.

[81] John E. Mitchell, Panos M. Pardalos, and Mauricio G. C. Resende. Interior point methods for combinatorial optimization. In *Handbook of Combinatorial Optimization*, pages 189–297. 1998.

[82] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. volume 13, pages 103–130, 1993.

[83] Roger Myerson. Refinements of the Nash equilibrium concept. *International Journal of Game Theory*, 7(2):73–80, 1978.

[84] Roger Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1991.

[85] Sylvia Nasar. *A Beautiful Mind*. Simon & Schuster, 1998.

[86] John F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*, pages 48–49, 1950.

[87] Sriraam Natarajan and Prasad Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 601–608, 2005.

[88] Gergely Neu and Csaba Szepesvari. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 2007.

[89] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670, 2000.

[90] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287, 1999.

[91] Wlodzimierz Ogryczak and Tomasz Sliwinski. On direct methods for lexicographic min-max optimization. In *Proceedings of the International Conference on Computational Science and its Applications*, pages 802–811, 2006.

[92] Guillermo Owen. *Game Theory*. Academic Press, 1995.

[93] Jing Peng and Ronald J. Williams. Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 1(4):437–454, 1993.

[94] Olivier Pietquin. *A framework for unsupervised learning of dialogue strategies*. PhD thesis, Faculty of Engineering, Mons (TCTS Lab), Belgium, 2004.

[95] Jos A. M. Potters and Stef H. Tijs. The nucleolus of a matrix game and other nucleoli. *Mathematics of Operations Research*, 17(1):164–174, 1992.

[96] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2007.

[97] Bob Price and Craig Boutilier. Implicit imitation in multiagent reinforcement learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 325–334, 1999.

[98] Bob Price and Craig Boutilier. Imitation and reinforcement learning in agents with heterogeneous actions. In *Proceedings of the Fourteenth Biennial Conference of the Canadian Society on Computational Studies of Intelligence*, pages 111–120, 2001.

[99] Bob Price and Craig Boutilier. A Bayesian approach to imitation in reinforcement learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 712–717, 2003.

[100] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[101] Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

[102] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Readings in speech recognition*, pages 267–296. 1990.

[103] Jette Randlov and Preben Alstrom. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471, 1998.

[104] Nathan Ratliff, David Bradley, J. Andrew Bagnell, and Joel Chestnutt. Boosting structured prediction for imitation learning. In *Advances in Neural Information Processing Systems 21*, pages 1153–1160, 2007.

[105] Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. Maximum margin planning. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 729–736, 2006.

[106] Ioannis Rexakis and Michail G. Lagoudakis. Classifier-based policy representation. In *Proceedings of the Seventh International Conference on Machine Learning and Applications*, pages 91–98, 2008.

[107] Lev Reyzin and Robert E. Schapire. How boosting the margin can also boost classifier complexity. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 753–760, 2006.

[108] Julia Robinson. An iterative method of solving a game. *The Annals of Mathematics*, 54(2):296–301, 1951.

[109] John Roycroft. *Test tube chess: A comprehensive introduction to the chess endgame study*. Stackpole Books, 1972.

[110] Walter Rudin. *Principle of Mathematical Analysis*. McGraw-Hill, 1976.

[111] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, Cambridge University Engineering Department, 1994.

[112] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 385–393, 1992.

[113] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[114] Jost Schatzmann, Kallirroi Georgila, and Steve J. Young. Quantitative evaluation of user simulation techniques for spoken dialogue systems. In *Proceedings of the Sixth SIGdial Workshop on Discourse and Dialogue*, pages 178–181, 2005.

[115] Jost Schatzmann, Blaise Thomson, and Steve J. Young. Statistical user simulation with a hidden agenda. In *Proceedings of the Eighth SIGdial Workshop on Discourse and Dialogue*, pages 273–282, 2007.

[116] Ulrich Schwalbe and Paul Walker. Zermelo and the early history of game theory. *Games and Economic Behavior*, 34(1):123–137, 2001.

[117] John Shawe-Taylor and Nello Cristianini. Margin distribution bounds on generalization. In *Proceedings of the Fourth European Conference on Computational Learning Theory*, pages 263–273, 1999.

[118] John Shawe-Taylor and Nello Cristianini. Further results on the margin distribution. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 278–285, 1999.

[119] Aaron Shon, David Grimes, Chris Baker, and Rajesh Rao. A probabilistic framework for model-based imitation learning. In *Proceedings of the Twenty-Sixth Annual Conference of the Cognitive Science Society*, pages 1237–1242, 2004.

[120] Satinder Singh, Michael Kearns, Diane Litman, and Marilyn Walker. Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems 14*, pages 956–962, 2000.

[121] Joe Smith. *Georgia Chess*, page 37, January 2008.

[122] Gilles Stoltz and Gabor Lugosi. Learning correlated equilibria in games with compact sets of strategies. *Games and Economic Behavior*, 59(1):187–208, 2007.

[123] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 881–888, 2006.

[124] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[125] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[126] Umar Syed and Robert E. Schapire. Imitation learning with a value-based prior. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 384–391, 2008.

[127] Umar Syed and Robert E. Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in Neural Information Processing Systems 22*, pages 1449–1456, 2008.

[128] Umar Syed and Robert E. Schapire. Apprenticeship learning using linear programming. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pages 1032–1039, 2008.

[129] Umar Syed and Jason D. Williams. Using automatically transcribed dialogs to learn user models in a spoken dialog system. In *Proceedings of the Forty-Sixth Annual Meeting of the Association for Computational Linguistics*, pages 121–124, 2008.

[130] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. In *Advances in Neural Information Processing Systems 18*, pages 25–32, 2004.

[131] Blaise Thomson, Jost Schatzmann, Karl Welhammer, Hui Ye, and Steve J. Young. Training a real-world POMDP-based dialog system. In *Proceedings of the Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technologies, Annual Meeting of the Association for Computational Linguistics*, pages 9–17, 2007.

[132] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the Twenty-First International Conference on Machine Learning*, page 104, 2004.

[133] Eric van Damme. *Refinements of the Nash equilibrium concept*. Lecture notes in economics and mathematical systems 219. Springer-Verlag, 1983.

[134] Eric van Damme. *Stability and perfection of Nash equilibria*. Lecture notes in economics and mathematical systems 219. Springer-Verlag, 1991.

[135] Vladimir N. Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16(2):264–280, 1971.

[136] Deepak Verma and Rajesh Rao. Imitation learning using graphical models. In *Proceedings of the Eighteenth European Conference on Machine Learning*, pages 757–764, 2007.

[137] John von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[138] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[139] Vladimir Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383, 1990.

[140] Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. Evaluating spoken dialogue agents with PARADISE: Two case studies. *Computer Speech and Language*, 12(4):317–341, 1998.

[141] Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Stable dual dynamic programming. In *Advances in Neural Information Processing Systems 22*, pages 1569–1576, 2008.

[142] Chris Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[143] Christopher J. C. H. Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[144] Marco A. Wiering and Edwin D. de Jong. Computing optimal stationary policies for multi-objective Markov decision processes. In *Proceedings of IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 158–165, 2007.

[145] Jason D. Williams and Steve J. Young. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(2): 393–422, 2007.

[146] C. F. Jeff Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[147] Bianca Zadrozny, John Langford, and Naoki Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the Third IEEE International Conference on Data Mining*, pages 435–442, 2003.

[148] Willard Zangwill. *Nonlinear Programming: A Unified Approach*. Prentice Hall, Inc., 1969.

[149] Ernst Zermelo. Über eine anwendung der mengenlehre auf die theorie des schachspiels. In *Proceedings of the Fifth Congress of Mathematicians*, pages 501–504, 1913.

[150] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient descent. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 928–936, 2003.

[151] Martin Zinkevich, Michael Bowling, and Neil Burch. A new algorithm for gener-

ating equilibria in massive zero-sum games. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, pages 788–793, 2007.

[152] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 22*, pages 1729–1736, 2008.