On the Sample Complexity of Reinforcement Learning

Sham Machandranath Kakade

Gatsby Computational Neuroscience Unit

University College London

PhD Thesis

March 2003

Abstract

This thesis is a detailed investigation into the following question: *how much data must an agent collect in order to perform "reinforcement learning" successfully?* This question is analogous to the classical issue of the *sample complexity* in supervised learning, but is harder because of the increased realism of the reinforcement learning setting. This thesis summarizes recent sample complexity results in the reinforcement learning literature and builds on these results to provide novel algorithms with strong performance guarantees.

We focus on a variety of reasonable performance criteria and *sampling models* by which agents may access the environment. For instance, in a policy search setting, we consider the problem of how much simulated experience is required to reliably choose a "good" policy among a restricted class of policies Π (as in Kearns, Mansour, and Ng [2000]). In a more online setting, we consider the case in which an agent is placed in an environment and must follow one unbroken chain of experience with no access to "offline" simulation (as in Kearns and Singh [1998]).

We build on the sample based algorithms suggested by Kearns, Mansour, and Ng [2000]. Their sample complexity bounds have *no dependence* on the size of the state space, an *exponential* dependence on the planning horizon time, and linear dependence on the complexity of Π. We suggest novel algorithms with more restricted guarantees whose sample complexities are again independent of the size of the state space and depend linearly on the complexity of the policy class Π, but have only a *polynomial* dependence on the horizon time. We pay particular attention to the tradeoffs made by such algorithms.

Acknowledgments

Many thanks to my family — Mom, Dad, Mish, and Suj — for all the love and encouragement you have given me. You have always taken keen interest and much enjoyment in my education and my life.

I also express deep gratitude to my advisor Peter Dayan for his guidance. I owe much to him for the clarity he has brought to my ideas and the freedom he gave me to pursue them.

I give warm thanks to John Langford, my closest collaborator in this work. Most work presented in this thesis was in direct collaboration with John or affected by his ideas. Only Kakade and Langford [2002] has been previously published, and this work appears mainly in chapter 7. John directly worked with me on the following results: the upper and lower bounds in chapter 2, the variance analysis for gradient methods, the CPI algorithm, and in providing the tightened upper and lower bounds of exploration in part 3. All other results in this thesis are original.

There are numerous other people I wish to thank, and it is likely I will forget someone but here goes. Matt Beal, Nathaniel Daw, Zoubin Ghahramani, Geoff Hinton, Sam Roweis, Maneesh Sahani, Yee-Whye Teh, Emo Todorov, and Chris Watkins strongly influenced my early ideas at the Gatsby Unit. I have also had edifying conversations with Peter Bartlett, Jonathan Baxter, Drew Bagnell, Daniela de Farias, Michael Kearns, Michael Littman, David McAllester, Andrew Ng, Satinder Singh, Dale Schuurmans, Rich Sutton, and Ben Van Roy.

I am grateful to my committee, Satinder Singh and Chris Watkins, for their feedback.

Finally, I must thank all my close friends with whom I have shared many good times during graduate school. I also thank all those friends who have visited me during my time in London (which I think includes just about of all of them).

I received financial support from the Gatsby Unit and the National Science Foundation.

Contents

CHAPTER 1

Introduction

Reinforcement learning has become the standard framework in the artificial intelligence community for studying how agents learn and plan in uncertain environments. In a reinforcement learning problem, an agent must learn a course of actions, *ie* a *policy*, through its interaction with a dynamic environment. Typically, the goal of an agent is to find or execute a policy that maximizes some measure of the long-term future reward. This paradigm is attractive because it offers a compact formalization of a host of problems that both people and artificial systems face.

Reinforcement learning is one step more realistic than the more widely studied problem of supervised learning. In supervised learning, the learner receives a "training set" $\{(x, y)\}$ of input/output pairs, where the output value y of an input x is a (possibly noisy) estimate of a "target function" $f(x)$. Usually, the samples in the training set are identically and independently distributed (*i.i.d.)* according to some distribution $P(x, y)$, and the goal of the learner (as in Valiant [1984]) is to construct an approximation to the relationship between the inputs and outputs such that when given a "test set" of input points distributed according to $P(x)$, the algorithm labels these points accurately.

In the past decade, much work in the field of learning has advanced our understanding of *efficient* supervised learning (see Anthony and Bartlet [1999] and Kearns and Vazirani [1994]). The quantities of interest are both the *computational complexity* and the *sample complexity* of finding a good approximation to the target function. Loosely, the sample complexity is: how large a training set is required in order to learn a good approximation to the target concept? The relevant computational complexity is: how much computation is required to manipulate a training set and output an approximation to the target?

The greater step toward realism in reinforcement learning stems from allowing the actions taken by an agent to affect the environment. This makes studying *efficiency* considerably harder for reinforcement learning than for supervised learning for various reasons. First, the environment doesn't unilaterally provide a "training set" to the agent. In general, the information the agent receives about the environment is determined by both the actions it takes and dynamics of the environment. Second, the information the agent receives is "partially labeled" in the sense that although the agent desires to maximize some measure of its long-term future reward, it only observes an immediate reward. Additionally, there is no sharp boundary between a "training" and "test" phase. The time the agent spends trying to improve the policy often comes at the expense of utilizing this policy — this is often referred to as the *exploration/exploitation* tradeoff.

Perhaps the two most important questions in the study of efficient reinforcement learning are as follows. The question of sample complexity is: *how much data must we collect in order to achieve "learning"?* The corresponding question of computational complexity is: *how much computation is required in using this data to achieve "learning"?* This thesis is

detailed investigation into the former question on the *sample complexity of reinforcement learning* (although to a lesser degree computational issues are also investigated). In general, the answers provided strongly depend on how the agent can access the environment as well as the performance criterion used to judge the amount of learning. This thesis summarizes recent sample complexity results in the reinforcement learning literature and builds on these results to provide novel algorithms with strong performance guarantees.

1.1. Studying the Sample Complexity

Let us now discuss a framework for studying the efficient use of samples. An informal notion of the sample complexity, which is in terms of the number of observed samples provided by some *sampling model* for the environment, was first discussed in Kearns, Mansour, and Ng [2000] (though see Kearns and Singh [1999] and Kearns, Mansour, and Ng [1999]). The first subsection presents some idealized sampling models. Then we discuss what constitutes efficient use of samples.

Idealized Sampling Models. The most general model is the **online simulation model** in which the environment itself is the sampling model and the agent has neither "offline" simulation access to the environment nor recourse to "resets", where a "reset" is the ability to move back to some fixed start state. In this model, the agent must follow one unbroken chain of experience for some number of *decision epochs*. Here a decision epoch is just a timestep in which a state is observed and an action is taken, and so the number of decision epochs is equivalent to the amount of observed experience. This is the most challenging reinforcement learning setting. The notion of sample complexity we consider is inspired by that of the E^3 algorithm of Kearns and Singh [1998]. Informally, the question we consider is: at how many states is the agent "exploring" and not "exploiting"?

A considerably more powerful sampling model is the **generative model**, which was introduced by Kearns, Mansour, and Ng [1999]. This model is a simulator which provides sampling access to any state in the environment of our choosing. This model is a much stronger assumption than having only online access, but it is a much weaker assumption than having complete knowledge of the environment. In real applications, this turns out to be a natural assumption, such as the case in which we have a physical simulator of the environment or where our model is in the form of some compact Bayes net. Here, we are often interested in the number of calls to the generative model required to find or execute a good policy (as in Kearns, Mansour, and Ng [1999,2000]).

We also consider an intermediate setting in which we have access to a μ**-reset model,** which allows "resets" of the state to some state chosen according to a fixed distribution μ, but is otherwise an online simulation model (as in Kakade and Langford [2002]). This is a considerably weaker assumption than the generative model, since we cannot access any particular state of our choice. Here, we consider algorithms which explicitly use the distribution μ as a preferential measure under which to optimize the policy. These algorithms are similar to and inspired by supervised learning algorithms which minimize the error with respect to some input distribution $P(x)$. This simulation condition could be quite useful, particularly if the fixed distribution μ provides us with states at which it is important to optimize the performance. Again the question we consider is: how many observed transitions are required to obtain a "good" policy? In this setting, we consider a notion of "goodness" that is defined with respect to μ.

What Constitutes Efficient Use of Samples? We study the sample complexity as a function of the sampling model at our disposal and the performance criterion used. In particular, *we consider the sample complexity to be the number of calls to the sampling model required to satisfy a specified performance criterion*, and we are interested in how this scales with the relevant problem dependent parameters. In the reinforcement learning setting, the parameters are the size of the state space N, the size of the action space A, the number of decision epochs T (or, alternatively, a discount factor γ), and the variance of the reward function. In addition, this scaling is dependent on an accuracy parameter ε (which is with respect to the performance criteria used) and a certainty parameter δ. In a policy search setting, where we desire to find a "good" policy among some restricted policy class Π, the dependency on the complexity of a policy class Π is also relevant (as in Kearns, Mansour, and Ng [2000]).

This thesis reviews and presents a variety of algorithms which use particular sampling models to return or execute "ε-good" policies, with probability of error less than δ. We consider upper and lower bounds on the sample complexity of these algorithms in terms of the aforementioned quantities. Close attention is paid into understanding what tradeoffs are made by various algorithms and what is reasonable to expect based on these tradeoffs.

In the supervised learning setting, the theoretical guarantees of most algorithms have *no dependence* on the size (or dimensionality) of the input domain which is analogous to N in our setting. Note that the supervised learning problem is closely related to a degenerate reinforcement learning problem where $T = 1$ (or $\gamma = 0$).

In contrast, many reinforcement learning algorithms ($T \neq 1$) depend polynomially on N, which is acceptable if the environment has a small state space. Unfortunately, the state space in many realistic settings is prohibitively large or infinite. The most important topic in the reinforcement learning literature over the last decade has been on the construction of algorithms which scale to cope with large or infinite state spaces.

Kearns, Mansour, and Ng [1999,2000] present two learning algorithms with a sample complexity that has *no dependence* on the size of the state space N, but have *exponential dependence* on the horizon time T. These algorithms provide an important, yet harsh, tradeoff. These "sparse sampling" methods call the generative model sufficiently many times such that a good policy can be computed or executed, but, in general, the samples obtained are insufficient to construct an accurate model of the environment (due to the lack of N dependence).

Particular attention is paid to the case of large or infinite state spaces and large horizon times. The most practically important novel algorithms provided are those with guarantees that have a *polynomial* dependence on T, and yet have *no dependence* on the size of the state space (along with a linear dependence on the complexity of the policy class Π). Understanding the tradeoffs made by such algorithms is perhaps the most practically relevant contribution of this work. Although the case of large action spaces is also important, this work does not focus on dealing with this setting though it is an important direction for further work (and we return to this point in the discussion of this thesis).

1.2. Why do we we care about the sample complexity?

Unlike in supervised learning, there is as of yet no "formal" definition in the literature of the sample complexity of reinforcement learning, though an informal one was provided in

Kearns, Mansour, and Ng [2000]. The cautious reader should ask: is the notion of sample complexity even relevant to the reinforcement learning setting?

Let us consider the two settings in which reinforcement learning is performed. One setting is where the agent has real ignorance about the environment, and samples are useful in an information theoretic sense (as in supervised learning). It is obvious that the notion of sample complexity is important for this case. In an alternative setting, the agent may have complete knowledge of the environment. This setting does not have an analogue in the supervised learning setting, since if the target function is known then our problem is solved. For this latter case, the agent only has computational ignorance about the world. Here, for computational purposes, our algorithm might simulate the environment, and the sample complexity can be viewed as a surrogate for the computational complexity.

Let us discuss these cases in turn, beginning with the complete knowledge setting.

Complete Knowledge of the Environment. The problem of finding a good policy in a fully known environment is perhaps the best studied problem in reinforcement learning. In some instances, the physical laws of the real world allow us to consider problems in which the environment dynamics are known. In other instances, the environment itself is artificially constructed with simple rules, such as in Chess, Tetris, Go, and Backgammon.

In large-scale problems where our knowledge of the environment is complete, it is rarely possible to specify a model of the environment in terms of a table of rewards and transition probabilities, and a compact model description is required. Commonly used representations of environment dynamics are systems of differential equations or generalizations of Bayes nets (*eg* dynamic Bayes nets or influence diagrams).

Using these compact models, it is often computationally expensive to perform certain exact computations, such as taking an expectation. However in a large class of compact models (such as Bayes nets), it is often computationally efficient to obtain Monte Carlo samples from a model and to use these samples for purposes of estimation.

Since Monte Carlo simulation is often the most tractable way to manipulate models, it is not surprising that most optimization techniques are simulation based. For these methods, a notion of "sample complexity" is how much experience must be simulated by our model in order to find a good policy. Note that for this complete knowledge setting, the "sample complexity" question is really a question of computational complexity, since to obtain a sample involves some amount of computation with our model. However, there is a natural split of the overall computational complexity into computations related to simulating the environment and computations related to optimization using these samples (such as in fitting a value function). Hence, the "sample complexity" provides a lower bound on the overall computational complexity (which is what we are ultimately interested in).

It should be noted that this notion of sample complexity is tied to using simulation based methods. Instead, if our algorithm could somehow directly manipulate the model (perhaps based on its special structure) to perform direct computations, then this notion is no longer relevant.

However, the two predominant techniques, value function methods and policy search methods, are simulation based. In simulation based value function methods, typically the policy is executed in the environment (using the model) to obtain sample trajectories and then some regression procedure is used to estimate its value (see Bertsekas and Tsitsiklis [1996] and Sutton and Barto [1998] for a thorough discussion of these methods). These values are

then used for policy improvement. By contrast, "direct" policy search techniques use simulated experience to find a good policy among some restricted set of policies without using any value functions (such as in policy gradient methods, see Baxter and Bartlett [2001] for review).

Incomplete Knowledge of the Environment. In many real applications, the dynamics of the environment are unknown. Here, we are strongly limited by what access we have to our environment. In many practically successful applications, we often have "off-line" access to the environment. For instance, we might have a physical simulator of the system which allows us to obtain estimates of the value (or gradient) of a policy by executing our policy in this physical simulator. The same sample complexity notion of the last subsection applies — though now the samples are "real" and not "computed" (*ie* information theoretically the samples provides us with more information about the environment).

Alternatively, we could attempt to construct a model by using sampled transitions in our physical simulator. This model could then be used for planning purposes to obtain a good policy for the task at hand. For example, in the (real) autonomous helicopter control problem (of Bagnell and Schneider [2001]), data was collected using a pilot tele-controlling the helicopter and using this data a non-parametric model of the dynamics was constructed. Importantly, due to the pilot's expertise, they had the ability to obtain samples in various regimes, which would otherwise have not been possible. This model was then used for planning purposes.

For this case, a notion of the sample complexity is how much experience from our physical simulator do we need to "accurately" construct a model of the environment. Here, what constitutes "accurate" is determined by what the model is used for. For instance, we might want a model that is minimally accurate enough to determine a good policy.

In the purest reinforcement learning setting, an agent is placed in an environment, with only limited knowledge of the environment and no "offline" simulation access. This is the most challenging setting, since the agent only obtains additional information through the actions it takes and must cope with any youthful mistakes it makes during the course of learning. In the previous setting, we only discussed efficiently obtaining a good policy. This is often a sensible goal when we have "offline" access to the environment or when there is a certain "learning" period in which poor performance by the agent is acceptable. In the "online" setting, we often care about maximizing some measure of the sum total reward that we obtain over some (possibly infinite) horizon.

1.3. Overview

This thesis focuses on Markov Decision Processes (MDPs) and is divided into three parts. Part 1 reviews the most commonly used approximate methods in the reinforcement learning community. It focuses on understanding why many of these methods do not enjoy strong performance guarantees (typically, performance guarantees depend on the size of the state space). This analysis is useful for motivating new algorithms with stronger performance guarantees. Part 2 is concerned with "sample-based" planning. The classical assumption for planning is that the agent has complete knowledge of the environment. Here, we consider the more reasonable case in which our planning algorithm has access to either a generative model or a μ-reset model. The policy search algorithms that are reviewed or presented here also have extensions in the partially observable (PO)MDP setting, and we

return to this point in the discussion of this thesis. In part 3, the unadulterated scenario is considered, in which the agent only has access to the online simulation model.

Of particular interest throughout this thesis is the use of *non-stationary, ie* time dependent, policies to optimize the future reward. The use of non-stationary policies leads to particularly clear results and a deeper understanding of the difficulty of planning in the reinforcement learning problem. The reason for this is that the use of non-stationary policies allows us to view the planning problem as a sequence of T supervised learning problems where the solution to each supervised learning problem is used to construct part of the non-stationary policy. Chapter 7 also considers the more challenging tricky problem of constructing a good *stationary* policy.

Part 1: Current Methods. Chapter 2 presents the standard definitions and the sampling models considered in this thesis. In addition to reviewing the exact algorithms (which assume complete knowledge of the MDP), this chapter also reviews generic planning algorithms which assume access to a generative model. The phased value iteration (similar to that in Kearns and Singh [1999]) uses the generative model to output a near-optimal policy and has a linear dependence on N and a polynomial dependence on T. The sparse sampling algorithm of Kearns, Mansour, and Ng [1999] *executes* a near-optimal policy and assumes access to the generative model during execution of the policy. This algorithm has *no dependence* on N but has an runtime dependence that is *exponential* in T. Lower bounds are also presented for both of these algorithms.

Chapter 3 reviews the standard approximate value function methods. Performance bounds are presented in terms of the intractable *max norm* regression error, which is a worst case error over the entire state space. This metric is the bane for obtaining strong sample complexity results independent of N. Typically, supervised learning algorithms (and the related theoretical analyses) exploit the fact that an *expectation* of a (bounded) random variable can be accurately obtained using a number of samples that has *no dependence* on the size of the input domain (this number depends only on an accuracy parameter ε, a certainty parameter δ, and the upper bound of the random variable). Exploiting this elementary sampling result in the reinforcement learning setting to provide algorithms with no dependence on N has proved to be quite elusive. This is often due to the *max norm* error not being an *expected* quantity.

This chapter also presents *convergence rates* for these approximate dynamic programming iterative methods, which are developed based on the analysis in Bertsekas and Tsitsiklis [1996]. Interestingly, these convergence rates are similar to those of their exact counterparts (though the regions to which these methods converge are obviously different). Additionally, this chapter briefly reviews the recent and promising approximate linear programming method of de Farias and Van Roy [2001], where the algorithm constructs an "accurate" approximation to the optimal value function (in an average l_1 sense). de Farias and Van Roy [2001] also have examined the sample complexity of this approach.

Chapter 4 focuses on simulation based, gradient methods (as in Marbach and Tsitsiklis [2001] and Baxter and Bartlett [2001]). These methods have achieved recent popularity due to their performance improvement guarantees. However, this chapter presents an analysis showing how the lack of exploration in gradient methods leads to an unreasonably (and arbitrarily) large variance in the estimates of the gradient direction (as discussed in Kakade and Langford [2002]). Thus, their finite-sample size convergence guarantees are particularly weak (though asymptotically they converge to a local optima).

Part 2: "Sample Based" Planning. Chapter 5 is concerned with performance bounds that shed light on the difficulty of the reinforcement learning problem. These bounds are extensions of the bounds of Bertsekas [1987] and Singh and Yee [1994]. Importantly, the performance bounds presented here are not stated in terms of a max norm error, but instead are stated in terms of *expectations with respect to the future state distribution* of an optimal policy. Informally, the future state distribution is a distribution over the state space induced by the state visitation frequency of a policy over the relevant horizon time. The bounds presented show how the reinforcement learning problem can be viewed as a supervised learning problem where the agent is "tested" under a distribution imposed by the optimal policy. These results directly motivate the non-stationary approximate policy iteration (NAPI) algorithm, which is presented in this chapter.

Chapter 6 considers the setting in which we desire to find a policy that has good performance as compared to those policies in some (potentially infinite) policy class Π. First, the trajectory tree method of Kearns, Mansour, and Ng [2000] is reviewed. This algorithm assumes access to a generative model and has an *exponential* dependence on T, a linear dependence on the *complexity* of the policy class Π, and *no dependence* on the size of the (potentially infinite) state space. Inspired by practical considerations, the question that is then addressed is: what guarantees can be made if we desire *polynomial dependence* on T, in addition to having no dependence on the size of the state space and linear dependence on the complexity measure of Π? Here, we consider finding a good *non-stationary* policy based on Π and the algorithm presented assumes access to only the weaker μ-reset model. The tradeoff paid for obtaining a polynomial dependence on T is that we now have a more restricted optimality guarantee that is stated in terms of distribution μ (yet the sample complexity bounds are independent of μ).

Chapter 7 examines the same problem as that in the previous chapter, except now we desire a *stationary* policy. Obtaining a good stationary policy proves to be a much more challenging problem. The conservative policy iteration algorithm is presented (from Kakade and Langford [2002]), which resorts to using *stochastic*, stationary policies. Again, the sampling model required by this algorithm is the μ-reset model. The sample complexity bounds and performance guarantees of this algorithm are comparable to the one from the previous chapter (*ie* polynomial in T, performance guarantees that depend on μ, *etc.)*.

Interestingly, the μ-based planning algorithms presented in part 2 are not guaranteed to return policies which are *both* stationary and deterministic.

Part 3: Exploration. Chapter 8 considers the purest scenario where the agent has no access to resets and can only obtain information about the environment through its choice of actions. Bounds are provided on what can be construed as the sample complexity of exploration. The notion that is considered is inspired by the E^3 algorithm of Kearns and Singh [1998], where the performance guarantees of E^3 are stated in terms of "mixing times" for the undiscounted case and in terms of the quality of the output policy of E^3 for the discounted case. This chapter provides a more general guarantee that is not stated in terms of "mixing times" and that is more parsimonious for both the discounted and undiscounted case. The question addressed is at how many states is the algorithm's expected long-term reward (with respect to some fixed horizon time) not near-optimal, where each timestep corresponds to one transition in the environment. Informally, this question is asking: at how many timesteps is the agent "exploring" and not "exploiting"? The algorithm

and bounds presented are developed from Kearns and Singh [1998] and Brafman and Tennenholtz [2001] and considerably tightened results are presented. Nonetheless, the results presented here are still stated in terms of the size of state space. Perhaps rather intuitively, an upper bound on the "sample complexity of exploration" is $O(N^2A)$ (neglecting log and other relevant factors), which is the number of parameters required to specify the transition model in the MDP. Lower bounds are also presented.

Chapter 9 examines the issue of model building for exploration. The algorithm presented in the previous chapter explicitly builds an *accurate* model of the MDP (at least in some subset of the states). However, the results presented in Kearns and Singh [1999] show that if the agent has access to a generative model, then a near-optimal policy can be obtained using an *impoverished* model of the world. This raises the controversial question of whether or not the demand to build an accurate model is too stringent. Adding to this conundrum, the discrepancy between the lower and upper bound presented in the last chapter is essentially the difference between building an accurate model of the world and using an impoverished model. The analysis presented in this chapter examines the possibility of constructing a crude model for exploration (with lower sample complexity), using the techniques described in Kearns and Singh [1999]. Unfortunately, this analysis does not lead to tightened results and the gap between our lower and upper bound persists.

1.4. "Agnostic" Reinforcement Learning

Before we begin, a few comments are in order about the approach taken in this thesis. The framework in which we work closely resembles that of the probably approximately correct (PAC) and agnostic learning framework for supervised learning (as in Valiant [1984], Haussler [1992] and Kearns, Schapire, and Sellie [1994]). There are two assumptions that characterize this framework in supervised learning. First, the setting is "distribution free" in the sense that no assumptions are made with regards to the input distribution $P(x)$. Although the error of interest is defined with respect to $P(x)$, the sample complexity bounds are independent of $P(x)$. Second, no assumptions are made about the "true target" function being contained in the hypothesis set $\mathcal{H}$.

Let us now outline the connections. For the policy search setting where the goal is to find a "good" policy in some restricted policy class Π, we make no assumptions about the environment and Π (as in Kearns, Mansour, and Ng [2000]). We still could (and should) use our problem dependent priors in choosing Π. However, as in the supervised learning, the theoretical guarantees do not assume these priors are correct.

In the setting where a μ-reset model is considered, although optimality criteria are stated in terms of μ, no assumptions are made on μ and the sample complexity bounds do not depend on μ. Hence, with respect to μ, the sample complexity bounds presented could be considered to be "distribution free". For the exploration setting, no knowledge is assumed about the environment (as in Kearns and Singh [1998]).

Our motivation for adopting this setting is identical to that given in supervised learning — we wish to understand fundamental sample complexity issues without making strong problem dependent assumptions. The most important and sensible counterpart to this approach is the Bayesian framework.

The natural Bayesian setting for reinforcement learning is one in which we have some prior distribution Q over environments. Here, the agent is set in an environment that is sampled according to Q. As usual, the goal of the agent is maximize some measure of expected

future reward, and for this case, the expectation is taken with respect to Q and the agent's course of actions. When working in this setting, it is important to think carefully about prior distributions Q over environments that are indicative of those that arise in practice.

Note that in this setting we assume complete knowledge of Q, so the problem is purely computational and can be cast as a POMDP whose adverse computational costs are well understood (see Littman [1996]). For a single state MDP, an optimal efficient algorithm exists using Gittins indexes (Gittins [1989]). We return to this case in the discussion and point out how the methods discussed herein have connections.

For situations in which the environment is fully known, more thought must be given to the Bayesian setting as to what constitutes appropriate priors. The reason being is that from an information theoretic perspective the agent has complete knowledge, and the problem is a purely computational one. Although, in solving the computational problem, we may invoke sampling methods, the issue of how to incorporate a Bayesian prior when doing this optimization requires more thought.

Part 1

Current Methods

CHAPTER 2

Fundamentals of Markov Decision Processes

The Markov decision process (MDP) is the model used throughout this thesis. This chapter reviews this framework along with the standard exact dynamic programming algorithms for MDPs. Special attention is paid to *non-stationary* policies, since the use of such policies leads to algorithms with strong performance guarantees. These algorithms are presented in chapters 5 and 6.

Fundamental to this thesis is the notion of a sampling model for the MDP. These sampling models are the means by which an agent obtains information about the MDP. As discussed in the introduction, the quantity of interest is how many calls to the sampling model are made by an algorithm in order to satisfy various performance criteria.

This chapter also introduces the sampling models used throughout this thesis and reviews two *generic*, near-optimal, "sample-based" planning algorithms, which assume access to a generative model (a natural simulator of the MDP). The first algorithm presented is phased value iteration which can be viewed as a sample based counterpart to the exact dynamic programming algorithms. A variant of this algorithm was originally developed by Kearns and Singh [1999] in order to analyze the Q-learning algorithm of Watkins [1989]. A slightly tightened sample complexity bound (as compared to Kearns and Singh [1999]) is provided on how many samples are required in order for the algorithm to compute a near-optimal policy. Interestingly, the reasons behind this tightened bound are related to the use of non-stationary policies. Lower bounds are also provided for this algorithm.

The second generic, "sample-based" algorithm reviewed is the sparse sampling algorithm of Kearns, Mansour, and Ng [1999]. Whereas phased value iteration returns a policy, the sparse sampling algorithm only returns a single action when given a state as input. In this sense, the algorithm itself acts a policy which uses the generative model at runtime. This algorithm executes a near-optimal policy and provides a different sample complexity tradeoff, since the the number of samples used by algorithm per call has no dependence on the size of the state space, but has an exponential dependence on the horizon time.

2.1. MDP Formulation

Consider the problem in which an agent is faced with the task of influencing an environment through the actions it takes. At each timestep the agent is at a state in the environment and it must make a decision of which action to perform. This action alters the state the agent is at and determines the reward the agent receives. The agent is allowed to make T such decisions. A Markov decision process formalizes this interaction between the agent and the environment.

DEFINITION 2.1.1. A **Markov Decision Process (MDP)** M is a tuple which consists of:

- A set of **decision epochs** $\{0, 1, \ldots, T-1\}$. This represents the set of times at which decisions are to be made. If T is finite, then the MDP is said to be a *finite horizon* MDP with T-epochs. If $T = \infty$, then the MDP is said to be an *infinite horizon* MDP.
- A set of **states** $\mathcal{S}$**.** This set is referred to as the *state space* and could be finite or infinite. If this state space is finite, the number of states is N.
- A set of **actions** $\mathcal{A}$**.** This set is assumed to be finite and of size A.
- The **transition model** $P(\cdot|s,a)$. For each $s \in \mathcal{S}$ and $a \in \mathcal{A}$, the probability distribution $P(\cdot|s,a)$ is on $\mathcal{S}$. The probability $P(s'|s,a)$ represents the probability of transitioning to s' after performing action a in state s.
- The **reward function** $r : \mathcal{S} \times \mathcal{A} \to [0,1]$. The reward function is always assumed to be deterministic and bounded such that $r(s,a) \in [0,1]$.

This treatment follows that of Puterman [1994], which should be referred to for a thorough definition of an MDP.

Both finite and infinite state spaces and both finite horizon and infinite horizon MDPs are treated in this thesis with respect to various optimality criteria. However, this thesis only considers MDPs which have a finite action set and which have a stationary transition model and a stationary reward function.[1]

Some comments regarding the technical assumptions on the reward function are in order. The assumption of a bounded reward function is necessary for finite time convergence results with sampling based methods. Our choice of $[0,1]$ as the bounded interval for r is for clarity of presentation, and the results provided easily generalize to the case of an arbitrary interval. The use of a deterministic reward function is for technical simplicity and it is straightforward to generalize the results to a (bounded) non-deterministic reward function.

A policy specifies a sequence of decision rules for action selection at all timesteps (or decision epochs) in M. For now, we only define *Markovian* (memoryless) policies. In part 3, we consider memory dependent policies, though these are termed *algorithms***.** The standard definitions of Markovian policies follow.

DEFINITION 2.1.2. Let M be a T-epoch MDP. A **policy** π is the sequence of distributions $\{\pi(\cdot|s,0), \pi(\cdot|s,1), \ldots, \pi(\cdot|s,T-1)\}$ where $\pi(\cdot|s,t)$ is a probability distribution on the action space. The probability $\pi(a|s,t)$ represents the probability of taking action a in state s at time t. A **deterministic policy** π is a policy in which each distribution $\pi(\cdot|s,t)$ is deterministic. We slightly abuse notation and write this policy as the function $\pi(s,t)$. A **stationary policy** π is a policy in which for every state s, the distribution $\pi(\cdot|s,t)$ does not change with time, and we write this distribution as $\pi(\cdot|s)$. A **deterministic stationary policy** π is a policy that is both deterministic and stationary, and again, we slightly abuse notation by writing this policy as the function $\pi(s)$.

Let us define a **path** as a sequence of state-actions, *eg* $(s_0, a_0, \ldots s_{T-1}, a_{T-1})$. A policy π for an MDP M along with a starting state s_0 induces a probability distribution over paths, where the probability of a path $(s_0, a_0, \ldots s_{T-1}, a_{T-1})$ is defined as:

$$\Pr(s_0, a_0, \ldots s_{T-1}, a_{T-1}|\pi, M, s_0) \equiv \pi(a_0|s_0,0)\Pi_{\tau=1}^{T-1} P(s_\tau|s_{\tau-1}, a_{\tau-1})\pi(a_\tau|s_\tau,\tau)\,.$$

[1]The "planning" methods in chapters 5 and 6 can be extended for finite horizon MDPs which have a time dependent transition model $P_t(\cdot|s,a)$ and reward function $r_t(s,a)$.

where P is the transition model of M. This distribution specifies the complete joint probability of state-action sequences in M under π from starting state s_0. Again, see Puterman [1994] for a thorough treatment of this induced stochastic process.

Under this distribution, the probability that the path $(s_t, a_t, \ldots s_T, a_T)$ is traversed in M from time t onward starting from state s_t at time t is then

$$\Pr(s_t, a_t, \ldots s_{T-1}, a_{T-1} | \pi, M, s_t) = \pi(a_t|s_t, t)\Pi_{\tau=t+1}^{T-1} P(s_\tau|s_{\tau-1}, a_{\tau-1})\pi(a_\tau|s_\tau, \tau)\,.$$

This latter distribution is useful when defining the value functions.

2.2. Optimality Criteria

The policy chosen by the agent induces a distribution over paths which in turn induces a distribution over the sequences of rewards the agent receives. The objective of the agent is to obtain a reward sequence that is as "large" as possible. This section defines some standard optimality criteria.

This thesis only treats the cases of maximizing the sum undiscounted reward in the finite horizon setting or maximizing the discounted future reward in an infinite horizon setting. This thesis does not consider maximizing the average reward in an infinite horizon setting. However, through standard notions of "mixing", maximizing the average reward in an infinite horizon setting has strong connections to both the finite horizon setting (see Kearns and Singh [1998]) and the discounted setting (see Baxter and Bartlett [2001] and Kakade [2001]).

This thesis breaks with tradition by only considering value functions which are *normalized* in both the discounted and undiscounted setting. This is for clarity of exposition and we return to this point in this section.

2.2.1. The Undiscounted Setting. The *normalized* undiscounted value of interest in the finite horizon setting is defined as follows.

DEFINITION 2.2.1. Let M be a T-epoch MDP and π be a policy with respect to M. The **value function** $V_{\pi,M}(s)$ for a state s is

$$V_{\pi,M}(s) \equiv \frac{1}{T} E_{(s_0, a_0, \ldots s_{T-1}, a_{T-1}) \sim \Pr(\cdot|\pi, M, s_0 = s)} \left[\sum_{\tau=0}^{T-1} r(s_\tau, a_\tau) \right] .$$

Note that this value is bounded in $[0, 1]$.

It is also convenient to consider the value of the reward obtained from time t onward. We term this the t-value, and it is defined as follows. We slightly abuse notation and use V to define this function.

DEFINITION 2.2.2. Let M be a T-epoch MDP, π be a policy with respect to M, and t be a timestep in M. The **t-value function** $V_{\pi,t,M}(s)$ for a state s is

$$V_{\pi,t,M}(s) \equiv \frac{1}{T} E_{(s_t, a_t, \ldots s_{T-1}, a_{T-1}) \sim \Pr(\cdot|\pi, M, s_t = s)} \left[\sum_{\tau=t}^{T-1} r(s_\tau, a_\tau) \right] .$$

We drop the M subscripts when the MDP is clear from context. Due to the factor of $\frac{1}{T}$, the function $V_{\pi,t}$ is bounded in $[0, \frac{t}{T}]$. Clearly, $V_\pi = V_{\pi,0}$.

For a deterministic policy π, these functions satisfy the following relation

$$V_{\pi,t}(s) = \frac{1}{T} r(s, \pi(s,t)) + E_{s' \sim P(\cdot|s,\pi(s,t))} \left[V_{\pi,t+1}(s') \right] .$$

Note the presence of the $\frac{1}{T}$ factor.This relation implies an efficient procedure for computing V_π that avoids using the full joint distribution $\Pr(\cdot|\pi, M, s_0 = s)$. This procedure is the essence of dynamic programming.

Another useful definition is that of the state-action value.

DEFINITION 2.2.3. Let M be a T-epoch MDP, π be a policy with respect to M, and t be a timestep in M. The t **state-action value function** $Q_{\pi,t,M}(s,a)$ for a state-action (s,a) is

$$Q_{\pi,t,M}(s,a) \equiv \frac{1}{T} r(s,a) + E_{s' \sim P(\cdot|s,a)} \left[V_{\pi,t+1,M}(s') \right] .$$

It is clear that $V_{\pi,t}(s) = E_{a \sim \pi(\cdot|s,t)} \left[Q_{\pi,t}(s,a) \right]$.

2.2.2. The Infinite Horizon, Discounted Setting. We now consider the discounted optimality criteria for infinite horizon MDPs. Recall that an infinite horizon MDP is one in which $T = \infty$. Let us break from tradition by defining *normalized* discounted value functions.

DEFINITION 2.2.4. A **discount factor** γ is in the interval $[0, 1)$. Let M be an infinite horizon MDP, π be a policy with respect to M, and γ be a discount factor.

The γ**-discounted value function** $V_{\pi,\gamma,M}(s)$ for state s is

$$V_{\pi,\gamma,M}(s) \equiv (1-\gamma) E_{(s_1,a_1,s_2,a_2,\ldots) \sim \Pr(\cdot|\pi,M,s_0=s)} \left[\sum_{\tau=0}^{\infty} \gamma^\tau r(s_\tau, a_\tau) \right] .$$

The γ**-discounted state-action value function** $Q_{\pi,\gamma,M}(s,a)$ at state-action (s,a) is

$$Q_{\pi,\gamma,M}(s,a) \equiv (1-\gamma) r(s,a) + \gamma E_{s' \sim P(\cdot|s,a)} \left[V_{\pi,\gamma,M}(s') \right] .$$

See Puterman [1994] for a more technically precise definition of this value function with respect to the sequence of random variables distributed according to $\Pr(\cdot|\pi, M, s_0 = s)$. For the γ-discounted setting, it is not particularly useful to define the t-values, since we typically use stationary policies in the discounted setting.

As in the finite horizon setting, the subscript of M is suppressed when M is clear from context. For a deterministic, stationary policy π, these discounted value functions satisfy the following consistency equations:

$$V_{\pi,\gamma}(s) = (1-\gamma) r(s, \pi(s)) + \gamma E_{s' \sim P(\cdot|s,\pi(s))} \left[V_{\pi,\gamma}(s') \right] .$$

Note how the use of normalized value functions alters the form of this equation in comparison to the unnormalized version. Again, this consistency equation is at the heart of dynamic programming methods.

2.2.3. A Word on the Use of Normalized Value Functions. Due to the normalization, the value functions $V_{\pi,M}$ and $V_{\pi,\gamma,M}$ lie in the bounded interval $[0,1]$. The literature sometimes uses normalized value functions for the T-step case, but rarely uses normalized value functions in the γ discounted case. The importance of normalization stems from the fact that often we are interested in ε-accurate approximations to the value functions.

Let us consider the γ-discounted setting. In the unnormalized case, the value function is bounded by $\frac{1}{1-\gamma}$, and so demanding an ε-accurate value function is somewhat unnatural since as $\gamma \to 1$, the ratio between ε and the upper bound of $\frac{1}{1-\gamma}$ tends to 0. This leads to sample complexity results that contain excessive factors of $\frac{1}{1-\gamma}$ due to this more stringent, unnatural fractional accuracy demand. In the normalized setting, an ε-approximation to the value function is more interpretable and intuitive, because regardless of the γ, ε represents the fractional accuracy compared to an upper bound of 1. Hence, the use of normalized value functions leads to sample complexity statements that are more interpretable than their unnormalized counterparts.

2.2.4. Optimal Value Functions and Optimal Policies. The standard definitions of optimality in the undiscounted setting follow.

DEFINITION 2.2.5. Let M be a T-epoch MDP and let Π be the set of all policies with respect to M.

The **optimal undiscounted value function** $V^*_M(s)$ for a state s is

$$V^*_M(s) \equiv \sup_{\pi \in \Pi} V_{\pi,M}(s)\,.$$

The **optimal undiscounted t-value function** $V^*_{t,M}(s)$ for a state s is

$$V^*_{t,M}(s) \equiv \sup_{\pi \in \Pi} V_{\pi,t,M}(s)\,.$$

A policy π is an **undiscounted optimal policy** at state s if

$$V_{\pi,M}(s) = V^*_M(s)\,.$$

The definitions in the discounted setting are analogous.

DEFINITION 2.2.6. Let M be an infinite horizon MDP, Π be the set of all policies with respect to M, and γ be a discount factor.

The **γ-discounted optimal value function** $V^*_{\gamma,M}(s)$ for a state s is

$$V^*_{\gamma,M}(s) \equiv \sup_{\pi \in \Pi} V_{\pi,\gamma,M}(s)\,.$$

A policy π is a **γ-discounted optimal policy** at state s if

$$V_{\pi,\gamma,M}(s) = V^*_{\gamma,M}(s)\,.$$

The optimal value functions satisfy the following well-known Bellman equations (Bellman [1957]):

$$\begin{aligned} V^*_t(s) &= \max_{a \in \mathcal{A}} \left(\frac{1}{T} r(s,a) + E_{s' \sim P(\cdot|s,a)} \left[V^*_{t+1}(s') \right] \right) \\ V^*_\gamma(s) &= \max_{a \in \mathcal{A}} \left((1-\gamma) r(s,a) + \gamma E_{s' \sim P(\cdot|s,a)} \left[V^*_\gamma(s') \right] \right)\,. \end{aligned}$$

Note that in the T-epoch case, the optimal t-value function is written in terms of the $t+1$ optimal value function. It is clear that optimal deterministic policies must satisfy

$$\begin{aligned}\pi^*(s,t) &\in \arg\max_{a\in A}\left(\frac{1}{T}r(s,a)+E_{s'\sim P(\cdot|s,a)}\left[V^*_{t+1}(s')\right]\right)\\ \pi^*(s) &\in \arg\max_{a\in A}\left((1-\gamma)r(s,a)+\gamma E_{s'\sim P(\cdot|s,a)}\left[V^*_{\gamma}(s')\right]\right)\ .\end{aligned}$$

respectively. For the discounted case, optimal policies that are both deterministic and stationary exist. It is a well known fact that these π^* are *simultaneously* optimal from every state-time or state, respectively.

2.3. Exact Methods

Given complete knowledge of the MDP M, there is a variety of algorithms to compute an optimal value function (both exactly and approximately). The optimal (or near-optimal) policy is then just the corresponding "greedy" policy. This section reviews the dynamic programming algorithms of value and policy iteration for both the T-step and γ-discounted case.

2.3.1. Value Iteration. The undiscounted value iteration algorithm for a T-epoch MDP is shown in algorithm 1. The algorithm recursively computes the exact optimal value functions for $t = T-1,\ldots 0$. Using these value functions, the optimal deterministic policy is computed.

Algorithm 1 Undiscounted Value Iteration(M)

(1) Set $V^*_T(s) = 0$.
(2) For $t = T-1,\ldots 0$

$$\begin{aligned}V^*_t(s) &= \max_{a\in\mathcal{A}}\left(\frac{1}{T}r(s,a)+E_{s'\sim P(\cdot|s,a)}\left[V^*_{t+1}(s')\right]\right)\\ \pi^*(s,t) &= \arg\max_{a\in\mathcal{A}}\left(\frac{1}{T}r(s,a)+E_{s'\sim P(\cdot|s,a)}\left[V^*_{t+1}(s')\right]\right)\end{aligned}$$

(3) Return π^* and V^*_t

Discounted value iteration (shown in algorithm 2) is similar to the undiscounted version except now the algorithm keeps track of a *vector* $J_t \in \mathcal{R}^N$. Let B be the **backup operator** defined as

$$[BJ](s) \equiv \max_{a\in\mathcal{A}}((1-\gamma)r(s,a)+\gamma E_{s'\sim P(\cdot|s,a)}\left[J(s')\right])\ .$$

The iterative algorithm sets $J_t = BJ_{t-1}$ and is run for T' steps. The policy returned is greedy with respect to the final vector $J_{T'}$.

Let us now address the quality of the greedy policy based on $J_{T'}$. Define the **max norm** (or the l_∞ **norm**) of J as follows

$$||J||_\infty \equiv \max_s |J(s)|\ .$$

A standard result is the contraction property for vectors J and J'

$$||BJ - BJ'||_\infty \le \gamma||J-J'||_\infty$$

Algorithm 2 Discounted Value Iteration(M, γ, T')

(1) Set $J_0 = 0$.
(2) For $t = 1, 2, \ldots T'$

$$J_t = BJ_{t-1}$$

(3) Return the policy

$$\pi(s) = \arg\max_{a \in \mathcal{A}} \left((1-\gamma) r(s,a) + \gamma E_{s' \sim P(\cdot|s,a)} \left[J_{T'}(s') \right] \right)$$

which implies

$$\begin{aligned} ||BJ_{T'} - J_{T'}||_\infty &\leq \gamma^{T'} ||BJ_0 - J_0||_\infty \\ &\leq (1-\gamma)\gamma^{T'} \end{aligned}$$

where the last line follows since $J_0 = 0$ and $||BJ_0||_\infty \leq (1-\gamma)$ due to our use of normalized reward functions. It can be shown that the greedy policy π based on this $J_{T'}$ satisfies, for all s,

$$V_{\pi,\gamma}(s) \geq V^*_\gamma(s) - 2\gamma^{T'}$$

(see Puterman [1994]).

2.3.2. Policy Iteration. In the exact setting, policy iteration is only defined in the γ-discounted case. For the undiscounted case, the policy iteration variant is identical to undiscounted value iteration.

Algorithm 3 presents policy iteration for the discounted case. The iterative algorithm constructs a policy π_t that is greedy with respect to the vector J_t , and the next vector J_{t+1} is just the value of the policy π_t.

Algorithm 3 γ-Discounted Policy Iteration(M, γ, T')

(1) Set the initial policy π_0 randomly.
(2) For $t = 1, \ldots T'$

$$\begin{aligned} J_t(s) &= V_{\pi_{t-1},\gamma}(s) \\ \pi_t(s) &= \arg\max_{a \in \mathcal{A}} \left((1-\gamma) r(s,a) + \gamma E_{s' \sim P(\cdot|s,a)} \left[J_t(s') \right] \right) \end{aligned}$$

(3) Return $\pi_{T'}$

Here, we have a slightly different contraction property (see Puterman [1994]),

$$\begin{aligned} ||V_{\pi_{T'},\gamma} - V^*_\gamma||_\infty &\leq \gamma ||V_{\pi_{T'-1},\gamma} - V^*_\gamma||_\infty \\ &\leq \gamma^{T'} ||V_{\pi_0,\gamma} - V^*_\gamma||_\infty \\ &\leq \gamma^{T'} \end{aligned}$$

where the last step follows since the value functions are normalized. Note that this contraction property is with the respect to the values of the policies themselves (unlike in value iteration which was with respect to the vector J_t).

2.3.3. Some Comments on the Choice of T'. These bounds on the convergence rate show that after T' updates both algorithms provide policies that are $O(\gamma^{T'})$ close to optimal.[2] Hence, if we choose

$$T' = \log_\gamma \varepsilon = O\left(\frac{-\log \varepsilon}{1-\gamma}\right)$$

then both algorithms provide policies that are $O(\varepsilon)$ near-optimal.

Perhaps unsurprisingly, this T' is just the time in which the finite sum of rewards $(1-\gamma)\sum_{t=0}^{T'-1}\gamma^t r_t$ is ε close to the infinite sum $(1-\gamma)\sum_{t=0}^{\infty}\gamma^t r_t$. This suggests that a non-stationary (T-epoch) version of value iteration also requires $O(\frac{\log \varepsilon}{1-\gamma})$ updates to find an ε-good non-stationary policy.

2.3.4. Other Methods. As shown by Williams and Baird [1993], there is a variety of asynchronous dynamic programming methods which interleave policy updates and value updates that converge to the optimal value function.

Additionally, linear programming can be used to compute V^* and this is the only known polynomial time algorithm for this exact computation (see Littman [1996] for a review of the complexity of these exact algorithms).[3] The exact optimal value function is specified as the solution to following linear program. For the discounted case, with variables $J(s)$,

$$\begin{aligned} \min_{J(s)} \quad & E_{s\sim\mu} J(s) \\ \text{s.t.} \; \forall s,a \quad & J(s) \geq (1-\gamma) r(s,a) + \gamma E_{s'\sim P(\cdot|s,a)}\left[J(s')\right] \end{aligned}$$

where μ is an any probability distribution that gives weight to all states.

2.4. Sampling Models and Sample Complexity

The classical assumption for planning is that the MDP is given explicitly by a table of rewards and transition probabilities. For large or infinite state MDPs, this assumption is clearly infeasible. Instead, of assuming complete knowledge of the MDP, this thesis considers various sampling models in which transitions based on the MDP can be observed by calling the sampling model. The sample complexity can be construed to be the number of calls to the sampling model required to achieve "learning". Clearly, this notion is dependent on the sampling model assumed and what constitutes "learning". As discussed in the introduction, the question of sample complexity is analogous to that in supervised learning, but significantly harder.

In the purest setting, we only assume access to an **online simulation model** of the MDP M. In this model, the agent is started at a state s_0 and the agent must follow a single unbroken chain of experience. In other words, the agent can take any action a and the next state is $s' \sim P(\cdot|s,a)$. The agent has no option to "reset" the MDP to another state. The need for explicit exploration is an important concern here. This is case is considered in part 3.

---

[2] However, these are just upper bounds on the algorithms. In practice, policy iteration appears to converge much faster than value iteration.

[3] To the authors knowledge, no exponential time lower bound on the computational complexity of the policy iteration algorithm exists where the algorithm operates in the *full* batch mode where $J \leftarrow BJ$. Also both exact value and policy iteration are polynomial time algorithms if the discount factor γ is fixed.

The following considerably stronger sampling model was introduced by Kearns, Mansour, and Ng [1999], defined as follows.

DEFINITION 2.4.1. A **generative model** $G(M)$ for an MDP M is a randomized algorithm that, on input of a state-action (s, a), outputs the reward $r(s, a)$ and a state s', where $s' \sim P(\cdot|s, a)$.

This model weakens the need for explicit exploration, since samples can be obtained from any state of our choice. Here, the issue of exploration is reduced to the problem of deciding which states to obtain samples from. Kearns and Singh [1998] and Kearns, Mansour, and Ng [1999,2000] show how this generative model can be used for near-optimal planning in a variety of settings. Two such methods are reviewed in the next section.

A weaker μ-**reset model** was introduced by Kakade and Langford [2002]. In this model, the agent has the option to reset the current state to a state s sampled according to μ, but the model is otherwise an online simulation model (the model is defined more formally in chapters 6 and 7). This model is considerably weaker than the generative model since it does not allow direct access to states of our choosing, but only allows us to "break" the chain of experience with a reset. The difficulty of exploration lies somewhere between that of the generative model and the online simulation model, since the agent has easy access only to states distributed according to μ.

An interesting situation arises if we have the ability to use a single μ-reset model of our choice. This choice potentially provides us with a natural means of incorporating prior domain knowledge. In chapters 6 and 7, a more refined notion of optimality is formulated in terms of the measure μ. As we shall see, a good choice of μ is one that matches the state visitation frequency of an optimal (or a near-optimal) policy. The choice of a measure over the state space is also particularly important to the recent approximate linear programming approach of de Farias and Van Roy [2001], where they also argue that domain knowledge is important in this choice.

2.5. Near-Optimal, "Sample Based" Planning

This section examines the sample complexity of two *generic*, near-optimal, sample-based planning algorithms which assume access to a generative model. The first algorithm we consider is phased value iteration, which is a sample-based version of exact value iteration. This algorithm returns a near-optimal policy. The second algorithm is the sparse sampling algorithm, which does not return a policy, but returns a single action when given a state as input. Here, the algorithm itself acts as a near-optimal policy, and the relevant sample complexity is that required to return a single action.

2.5.1. Phased Value Iteration. Clearly, with only access to a generative model $G(M)$ of an MDP M, exact value iteration is not feasible. Instead, one could consider obtaining samples from $G(M)$ to empirically perform the backups. The phased value iteration does just this.

Undiscounted phased value iteration is shown in algorithm 4 (which is a variant of phased Q-learning by Kearns and Singh [1999]). During each iteration t, the algorithm calls the generative model m times per state-action, so a total of mNA calls are made. The algorithm then uses these samples to construct an empirical model $\hat{P}_t$ of P and this empirical

model $\hat{P}_t$ is used to do the t-th backup. The total number of calls to the generative model made by the algorithm is $mNAT$.

Algorithm 4 Undiscounted Phased Value Iteration$(G(M), m)$

(1) Set $\hat{V}_T(s) = 0$
(2) For $t = T-1, \ldots 0$
  (a) Using m calls to $G(M)$ for *each* state-action

$$\hat{P}_t(s'|s,a) = \frac{\#\text{ of times } (s,a) \to s'}{m}$$

  (b) Set

$$\begin{aligned}\hat{V}_t(s) &= \max_{a\in\mathcal{A}} \left(\frac{1}{T} r(s,a) + E_{s'\sim\hat{P}_t(\cdot|s,a)} \left[\hat{V}_{t+1}(s') \right] \right) \\ \hat{\pi}(s,t) &= \arg\max_{a\in\mathcal{A}} \left(\frac{1}{T} r(s,a) + E_{s'\sim\hat{P}_t(\cdot|s,a)} \left[\hat{V}_{t+1}(s') \right] \right)\end{aligned}$$

(3) Return $\hat{\pi}$ and $\hat{V}_t$

The following theorem addresses how many observed transitions, using a generative model, are sufficient to compute a near-optimal policy from *every* state. This sample complexity bound was first addressed by Kearns and Singh [1999]. The result presented here provides an improved dependency in terms of T (which is due to the non-stationary algorithm).[4]

THEOREM 2.5.1. *(Upper Bound) For an appropriate choice of m, the phased value iteration algorithm calls the generative model $G(M)$*

$$O\left(\frac{NAT^3}{\varepsilon^2} \log \frac{NAT}{\delta} \right)$$

times and with probability greater than $1-\delta$, returns a policy π such that for all states s,

$$V_\pi(s) \geq V^*(s) - \varepsilon\,.$$

Importantly, note that this bound is linear in NA (neglecting log factors), which is significantly less than the N^2A number of parameters it takes to just specify the transition model of M.

The proof is based on the one in Kearns and Singh [1999]. The proof entails finding an appropriate value of m such that $\hat{V}_t$ is a good approximation to V_t^*. The key to the improved sample size result is in showing that $\hat{V}_t$ is a *also* a good approximation to the value of the greed policy, $V_{\pi,t}$. This latter fact is tied to the use of a non-stationary policy.

PROOF. Assume that the following expectations are ε accurate for all s, a, t:

$$\left| E_{s'\sim P(\cdot|s,a)} \left[\hat{V}_t(s') \right] - E_{s'\sim\hat{P}_{t-1}(\cdot|s,a)} \left[\hat{V}_t(s') \right] \right| \leq \varepsilon\,. \tag{2.5.1}$$

[4]Their analysis did not focus on the horizon time and treated γ as a constant. However, if we examine the complexity in terms of $H = \frac{1}{1-\gamma}$ then the bound is a factor of H^2 more than that presented here. The difference is due to the fact that our non-stationary algorithm allows to us show that $\hat{V}$ is a good approximation to V^*(see the proof). It is not clear how to prove this using a stationary policy.

Later, an appropriate value of m is chosen to satisfy this condition. It follows that

$$\begin{aligned}
|V^*_{t-1}(s) - \hat{V}_{t-1}(s)| &\leq \max_a |E_{s' \sim P(\cdot|s,a)} [V^*_t(s')] - E_{s' \sim \hat{P}_{t-1}(\cdot|s,a)} \left[\hat{V}_t(s')\right]| \\
&\leq \max_a |E_{s' \sim P(\cdot|s,a)} [V^*_t(s')] - E_{s' \sim P(\cdot|s,a)} \left[\hat{V}_t(s')\right]| + \varepsilon \\
&\leq \max_s |V^*_t(s) - \hat{V}_t(s)| + \varepsilon \, .
\end{aligned}$$

Recall that π is the greedy policy with respect to $\hat{V}_t$. Let $a = \pi(s, t-1)$, and so $\hat{V}_{t-1}(s) = \frac{1}{T} r(s,a) + E_{s' \sim \hat{P}_{t-1}(\cdot|s,a)} \left[\hat{V}_t(s')\right]$. Similarly,

$$\begin{aligned}
|V_{\pi,t-1}(s) - \hat{V}_{t-1}(s)| &\leq |E_{s' \sim P(\cdot|s,a)} [V_{\pi,t}(s')] - E_{s' \sim \hat{P}_{t-1}(\cdot|s,a)} \left[\hat{V}_t(s')\right]| \\
&\leq |E_{s' \sim P(\cdot|s,a)} [V_{\pi,t}(s')] - E_{s' \sim P(\cdot|s,a)} \left[\hat{V}_t(s')\right]| + \varepsilon \\
&\leq \max_s |V_{\pi,t}(s') - \hat{V}_t(s')| + \varepsilon \, .
\end{aligned}$$

Recursing on the previous two equations, leads to:

$$\begin{aligned}
\max_s |V^*(s) - \hat{V}(s)| &< \varepsilon T \\
\max_s |V_\pi(s) - \hat{V}(s)| &< \varepsilon T \, .
\end{aligned}$$

and so $\max_s |V_\pi(s) - V^*(s)| < 2\varepsilon T$.

It remains to choose m such that equation 2.5.1 holds with error $\frac{\varepsilon}{2T}$, which ensures that our policy will be ε near-optimal. Since each $\hat{P}_{t-1}$ is constructed independently of $\hat{V}_t$, we can apply Hoeffding's bound. There are NAT of these conditions that must hold, and so by Hoeffding's bound and union bound, we have the probability that equation 2.5.1 fails is less than $NAT \exp(-2\varepsilon^2 m / T^2)$. If we demand that this probability be less than δ, this implies $m = O(\frac{T^2}{\varepsilon^2} log \frac{NAT}{\delta})$. The result follows, since $mNAT$ calls to the generative model must be made. □

This phased algorithm is considered to be "direct" rather than "model based". This is because at each step t an independent batch of samples is obtained to do each backup. In contrast, in a "model based" approach, all the samples would be used to construct only one empirical model of the world, and this model would be used for planning purposes. The "direct" variant is considered here since, in this analysis, it provides a tighter sample size result over the model based approach, with respect to the horizon time (unlike the analysis in Kearns and Singh [1999]). The model based approach is considered in chapter 9.

The following lower bound shows that, in general, the factor of N in the upper bound cannot be reduced if instead we only demand to obtain a near-optimal policy from just a *single* state. This shows that the gap between the lower and upper bound is a factor of $\frac{T^2}{\varepsilon}$ (ignoring log factors).

THEOREM 2.5.2. *(Lower Bound) Let $\mathcal{A}$ be an algorithm that is given only access to a generative model for an MDP M, and inputs s, ε, and δ. Assume the output policy π satisfies, with probability greater than $1-\delta$, $V_\pi(s) \geq V^*(s) - \varepsilon$. There exists an MDP M and a state s, on which $\mathcal{A}$ must make $\Omega\left(\frac{NAT}{\varepsilon} \log \frac{1}{\delta}\right)$ calls to the generative model $G(M)$.*

This theorem uses the common Ω notation, where $f = \Omega(g)$ if $g = O(f)$. The proof is provided in the last subsection.

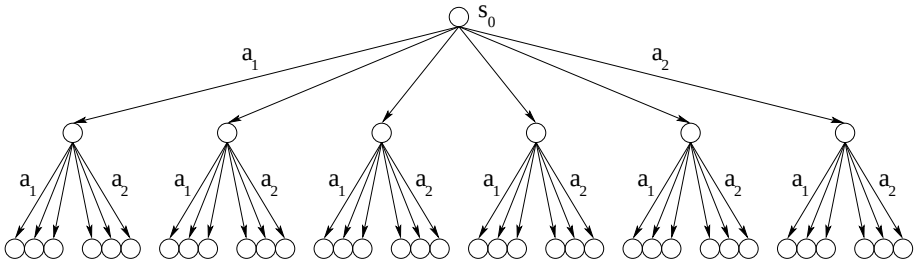

FIGURE 2.5.1. A sparse sample "look-ahead" tree constructed using a generative model with $A = 2$, $m = 3$, and $H_\varepsilon = 2$.

2.5.2. The Sparse Sampling Algorithm. The sparse sampling algorithm of Kearns, Mansour, and Ng [1999] takes a different "on-line" approach. In the approach described above, phased value iteration uses the generative model to construct a policy, and the policy returned is just a table of probabilities. For large or infinite MDPs, it is clear that storing a policy in a tabular representation is infeasible, let alone computing this policy. Instead, the sparse sampling algorithm implements the policy itself and the algorithm uses the generative model at each state to compute an action at that state. In this sense, the algorithm itself could be considered to be a compact representation of the policy.

A high level description of sparse sampling algorithm and the insight behind the proof is now provided. In the infinite horizon, γ-discounted setting, a cutoff time $H_\varepsilon = O(\frac{-\log \varepsilon}{1-\gamma})$ is imposed, which introduces a bias of ε into estimates of the discounted value function over this horizon (see subsection 2.3.3).

First, let us specify an algorithm for the simple, deterministic case. Start by using the generative model to do a brute force lookahead search, *ie* try every action once at every state reached until the depth H_ε is reached. This requires $O(A^{H_\varepsilon})$ calls to the generative model. After doing this, it is clear we have observed all possible outcomes until this depth, and dynamic programming suffices to compute a near-optimal policy from the root state.

For the general stochastic MDP case, the description of the algorithm/policy $\mathcal{A}$ is as follows. When $\mathcal{A}$ is given a single state s as an input, $\mathcal{A}(s)$ builds a tree with s as the root state. This tree is used to compute a *single* action, and $\mathcal{A}$ returns this single action. When $\mathcal{A}$ is viewed as a policy being executed, the algorithm/policy $\mathcal{A}$ builds a tree for each input state s and then executes the single output action $\mathcal{A}(s)$. The question is then: how should we build this tree such that the policy implemented by $\mathcal{A}$ is near-optimal? Clearly for an infinite state space, stochastic MDP M, it is not feasible to construct a tree which accurately approximates the transition model in M using only a generative model $G(M)$, unlike in the deterministic case. However, for $\mathcal{A}$ to be a near-optimal policy, $\mathcal{A}$ only needs to build *sparsely sampled* trees.

A tree can be built in the obvious way (as is shown in figure 2.5.1): at the root state, call the generative model m times for each action to create mA children (so there are m children for each action), and then recursively perform this procedure on each child until a depth of H_ε is reached. Label each node with the associated reward. This tree naturally induces an MDP M' in which nodes are the states and taking an action from a state causes a uniform transition to a child node (assume the leaves are absorbing). The *single* action returned by $\mathcal{A}$ at state s is just the optimal action on M' at the root state s. Hence, during the execution

of the policy $\mathcal{A}$, a tree must be constructed for each state s visited by the policy, which requires $O((mA)^{H_\varepsilon})$ calls to the generative model.

The following theorem shows that the size of the tree is *independent* of the size of the state space, yet the policy $\mathcal{A}$ is ε near-optimal. This is because m can be chosen to be polynomial in A, ε, and H_ε. The tradeoff is that the number of calls to generative model is *exponential* in the horizon H_ε for just *one* call to $\mathcal{A}$.

For comparison to T, define an analogous horizon time

$$H \equiv \frac{1}{1-\gamma}$$

and the theorem is stated in terms of H.

THEOREM 2.5.3. *(Sparse Sampling; Kearns, Mansour, and Ng [1999]) Let M be an MDP, and let $\mathcal{A}$ be a sparse sampling algorithm with access to the generative model $G(M)$. For an appropriate choice of m, the number of calls to the generative model $G(M)$ during each call to $\mathcal{A}$ is*

$$\left(\frac{AH}{\varepsilon}\right)^{O\left(H \log \frac{H}{\varepsilon}\right)}$$

Furthermore, the value function of the policy implemented by $\mathcal{A}$ satisfies

$$V_{\mathcal{A}}(s) \geq V^*(s) - \varepsilon\,.$$

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

Importantly, although the algorithm is sample based, there is no confidence parameter δ here. The *expectation* of the discounted return achieved by $\mathcal{A}$ is ε-near to the optimal value (with probability 1).

Interestingly, the tree MDP M' is, in general, a terrible approximation to M since the size of the tree has no dependence on the size of the state space. In fact, after executing any action a returned by the policy $\mathcal{A}$ the next state observed is in general a state that was *not* present in the tree that was constructed to choose this action (consider the case of a continuous state space). Contrast this to the deterministic case, where the tree provides a perfect model of the MDP up to depth H_ε.

Now the high-level intuition behind the proof is provided (see Kearns, Mansour, and Ng [1999] for the full proof). Let us consider the simpler problem of computing an approximation to the T-step optimal value function $V_0^*(s)$ at a particular state s for a binary action MDP. For now, assume that we know the function V_1^*. For each action a, let us call the generative model m times with (s,a) and construct the quantities $\frac{1}{T} r(s,a) + \sum_i V_1^*(s_i)$ for each action, where $\{s_i\}$ are the samples obtained from the generative model called with (s,a). Then an estimate of $V_0^*(s)$ is just the max of these quantities. It is straightforward to show that if we set $m = O(\frac{1}{\varepsilon^2} \log \frac{A}{\delta})$, then our estimate of $V_0^*(s)$ is ε accurate with error probability less than δ. Note that m does not depend on the size of the state space.

The two key insights to the analysis are as follows. The first is that we only need to know the values $V_1^*(s_i)$ at the sampled states s_i to approximate $V_0^*(s)$ and do not need to know the entire function V_1^*. The second is that if an ε' approximation is used for $V_1^*(s_i)$ instead of its exact value then our estimate of $V_0^*(s)$ is an $\varepsilon + \varepsilon'$ approximation. These points imply the recursive estimation procedure for $V_0^*(s)$ using the tree, where we *only* estimate the functions V_t^* at states in the tree. We start at the leaves, where V_T^* is 0. Then, recursively, we do the "backups" to estimate V_t^* with our estimate of V_{t+1}^* from depth

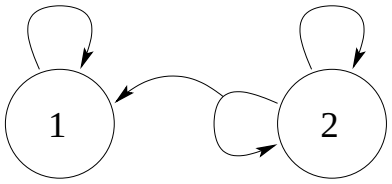

FIGURE 2.5.2. MDPs in which learning is difficult (for $A = 2$). See text for description.

$t + 1$. The proof carefully alters ε to account for the propagation of errors and δ to ensure the total error probability is appropriate.

Now let us return to the sparse sampling algorithm. The procedure for computing a near-optimal action at the root node s is a slight variation on the procedure described above. Essentially, we use the estimates of V_1^* to choose the best action at s rather than to estimate $V_0^*(s)$. The only caveat is that the certainty factor δ is not present in the statement of theorem 2.5.3. This certainty factor can be absorbed into the error ε (since an independent tree is built for every state visited during the execution of $\mathcal{A}$). See Kearns, Mansour, and Ng [1999] for complete details.

The following lower bound shows that in the worst case the exponential dependence on H is unavoidable.

THEOREM 2.5.4. *(Lower Bound; Kearns, Mansour, and Ng [1999]) Let $\mathcal{A}$ be an algorithm that is given access only to a generative model for an MDP M, and inputs state s and ε. Let the stochastic policy implemented by $\mathcal{A}$ satisfy $V_\pi(s) \geq V^*(s) - \varepsilon$ for all states s. Then there exists an MDP M on which $\mathcal{A}$ must make $\Omega\left((\frac{1}{\varepsilon})^{\frac{1}{\log \gamma}}\right)$ calls to the generative model $G(M)$.*

Note that for large γ, $\log \gamma = O(H)$, so this lower bound is approximately $(\frac{1}{\varepsilon})^H$. For completeness, the proof is provided in the next section.

2.5.3. Lower Bounds and "Challenging" MDPs. The proof for the lower bound on the sample complexity for returning an optimal policy from just a *single* state involves constructing a "well-mixed" MDP in which learning is difficult.

PROOF. (proof of theorem 2.5.2) First, let us consider a family of two state MDPs (see figure 2.5.2). The first state is an absorbing state with a maximal reward of 1. For the second state, there are A actions, all of which have 0 reward. Of these actions, $A-1$ of them lead to self transitions and the remaining action has an associated transition probability of $\frac{\varepsilon}{T}$ to the absorbing state. Label which action is this remaining action randomly.

The optimal value $V^*(2)$ is equal to $p(\text{escape from } 2)$ times the expected normalized reward assuming escape has occurred. The probability of escape in T steps is $\Omega(\varepsilon)$, since the probability of escape is $\frac{\varepsilon}{T} * T$ plus higher order terms. The normalized reward assuming that escape does occur is the fraction of the T-steps spent in state 1 *given* that escape has occurred. This is $\Omega(1)$. This makes $V^*(2) = \Omega(\varepsilon)$. Hence, the agent must discover this transition in order to execute a T-step policy which has expected return that is ε near-optimal, from state two.

The probability that the agent does not transition to the absorbing state from state two when the rewarding action is tried k times is $(1-p)^k$. Thus, in order to just discover this transition, with probability greater than δ, the number of calls to the generative model is $\frac{\log \delta}{\log(1-p)} = \Omega(\frac{1}{p} \log \delta)$. The algorithm must take every action this number of times, since a test of one action provides no information about another action. Thus, $\Omega(\frac{AT}{\varepsilon} \log \delta)$ is a lower bound on the number of calls to the generative model in order to obtain a near-optimal policy at state 2.

The extension to an N state MDPs is as follows. State 1 is identical to that above, and all other states are non-rewarding. At any state $i > 1$, $A-1$ of the actions transition uniformly to a non rewarding state. The remaining action has a probability of $\frac{\varepsilon}{T}$ of entering state 1, else the transition is uniform to a non-rewarding state. Hence, to act optimally for any single state $i > 1$, the agent must discover the rewarding action at $\Omega(N)$ of the states, since the agent is visiting these states uniformly before entering the rewarding state. Discovery at each state requires $\Omega(\frac{AT}{\varepsilon} \log \delta)$ calls, so the total number of calls is $\Omega(\frac{NAT}{\varepsilon} \log \delta)$. □

The proof of the lower bound for the sample complexity of the sparse sampling algorithm follows.

PROOF. (proof of theorem 2.5.4 from Kearns, Mansour, and Ng [1999]) Define an MDP based on a binary tree of depth $\log_\gamma \varepsilon$. The states are the nodes in the tree and the actions are $\{1, 2\}$. Action a at state s results in a transition to the a-th child of s. The leaves are absorbing. Choose a random leaf to be maximally rewarding and set the rewards at all other states to be 0. If $\mathcal{A}$ is given the root node of this tree, then $\Omega(2^H)$ calls to the generative model must be made in order to just discover the rewarding node. □

CHAPTER 3

Greedy Value Function Methods

The most widely-used techniques for obtaining approximate solutions to large-scale reinforcement learning problems are approximate value function methods. The basic idea is to approximate the value functions (or state-action values) with some regression algorithm and use these approximations in lieu of their counterparts in an exact method. Typically, the regression algorithm used is simulation based, where a "training set" is constructed by obtaining Monte Carlo estimates of the policy from various states. This has lead to a number of empirical successes including backgammon (Tesauro [1994]), job-shop scheduling (Zhang and Dietterich [1995]), dynamic channel allocation (Singh and Bertsekas [1997]) and chess (Baxter, Tridgell, and Weaver [2000]).

3.0.4. Background. There are a plethora of greedy approximate methods in the literature (see Sutton and Barto [1998], Bertsekas and Tsitsiklis [1996], and Gordon [1999]). The most straightforward of these are just approximate variants of value or policy iteration, where there are distinct policy update phases and value update phases. We review the performance guarantees of these methods in this chapter.

A variety of more asynchronous schemes are also commonly used, such as optimistic policy iteration, SARSA, Dyna-Q, etc. (see Sutton and Barto [1998], Bertsekas and Tsitsiklis [1996] and Singh [1994]). These methods interleave the policy updating and value updating, without waiting for convergence of the policy evaluation algorithm. Typically, the policy evaluation algorithm makes slow changes determined by a "learning rate" parameter, and the policy is greedy with respect to these values (or the policy is updated occasionally). Part of the reasoning behind these latter methods is to avoid making more drastic policy changes, which is often the problem in providing convergence results.

In fact, much of the literature has focused on obtaining various convergence results for these algorithms. In general, it is expected that "chattering" occurs for many algorithms, where the policy fluctuates between some set of policies without ever converging (see Gordon [1996] and Bertsekas and Tsitsiklis [1996]). Bertsekas and Tsitsiklis [1996] provide the most extensive convergence analysis (both experimental and theoretical) for a variety of algorithms. For TD-learning, convergence results have focused on the quality of the policy evaluation for a *single* policy (Tsitsiklis and Van Roy [1997]). Gordon [1995,2001] has also studied the convergence properties of a variety of algorithms, and has shown that SARSA(0) doesn't converge (but it converges to a region). Other negative results exists, such as the divergence of Q-learning with function approximation (Baird [1995]). There are a number of cases where significant policy degradation has been observed during the course of an algorithm (Boyan and Moore [1995], Weaver and Baxter [1999] and the Tetris example in Bertsekas and Tsitsiklis [1996]).

3.0.5. The Question of Sample Complexity. Convergence results are often a first step in obtaining more powerful results. Ultimately, the quantities of interest are the time it takes a planning algorithm to halt, the related sample complexity, and the quality of the output policy. Furthermore, it is not unreasonable to allow algorithms where the policy "chatters", provided that the set of policies in which the algorithm chatters around all have acceptable performance and that this asymptotic set is reached quickly.

Asymptotic convergence results do not shed light on the answers to these questions. In fact, in the limit of an infinite amount of data, we could argue that any sensible algorithm should find an optimal policy (at least if the MDP is finite). Additionally, many of the convergence results do not address the quality of the final policy returned by the algorithm and this question seems particularly difficult to address. Those bounds that do exist are typically stated in terms of a *max norm* error of the policy evaluation step, which is the *worst case* error over the entire state space.

This max norm error is the bane in providing strong sample complexity results for these approximate iterative methods that are independent of the size of the state space. Though asymptotically this error can be minimized within some parametric class of function approximators, finite sample size bounds are not well understood.

Furthermore, most algorithms typically do not directly minimize this max norm error. For example, the common error metric used in TD methods is the mean squared error under an "on-policy" distribution, *ie* a distribution that is induced by the state visitation frequency of the current policy (see Tsitsiklis and Van Roy [1997]). If such a function approximation scheme is used, say in approximate policy iteration, then it is unclear what the quality of the final policy will be (since it is the max norm which determines this latter quantity). A crude attempt to keep the max norm error small might use a somewhat more uniform distribution (for the mean squared error) in a TD method. However, the convergence properties of TD under an "off-policy" measure are not clear (Tsitsiklis and Van Roy [1997]).

This chapter focuses on the fundamental convergence results for approximate iterative algorithms based on the max norm error. Examples are provided which suggest that the max norm error is the appropriate error to consider for these algorithms. Although the policy itself does not converge, the quality of the asymptotic set of policies reached can be bounded in terms of the max norm error. Further, stronger results on the *convergence rate*, at which this asymptotic performance level is achieved, are stated. Most of the theorems in this chapter have been developed from the analysis in Bertsekas and Tsitsiklis [1996]. In addition, a promising recent linear programming approach of de Farias and Van Roy [2001] is also discussed (where sample complexity bounds of this approach have been explicitly examined).

3.1. Approximating the Optimal Value Function

For simplicity, this chapter only deals with finite state spaces and deterministic policies. We only work in the γ-discounted setting (so the γ subscripts are suppressed).

Define Q^* to be the optimal state-action value, *ie* $Q^*(s,a) \equiv Q_{\pi^*}(s,a)$ where π^* is an optimal policy. Let us start by assuming that we have an estimate $\tilde{Q}$ of Q^* and that the max norm error (or the l_∞ error) of $\tilde{Q}$ is bounded by ε, *ie*

$$||\tilde{Q} - Q^*||_\infty \leq \varepsilon$$

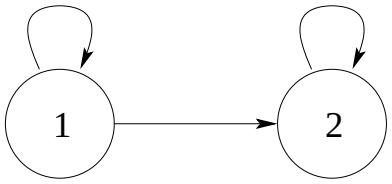

FIGURE 3.1.1. An example showing the bound in theorem 3.1.1 is tight. See text for description.

where $||x||_\infty = \max_{s,a} |x(s,a)|$ for $x \in \mathcal{R}^{N \times A}$. The standard procedure is to use the greedy policy $\pi(s) = \arg\max_{a \in \mathcal{A}} \tilde{Q}(s,a)$. The following theorem from (Bertsekas [1987] and Singh and Yee [1994]) bounds the quality of this policy.

THEOREM 3.1.1. *Assume $||\tilde{Q} - Q^*||_\infty \leq \varepsilon$ and let π be the greedy policy with respect to $\tilde{Q}$. Then for all states s,*

$$V_\pi(s) \geq V^*(s) - \frac{2\varepsilon}{1-\gamma}.$$

This shows that for a greedy update our policy does not get worse by more than a factor related to our worst case error ε.

PROOF. Let π^* be an optimal policy. By construction of π, $\tilde{Q}(s,\pi(s)) \geq \tilde{Q}(s,\pi^*(s))$. Using this and the approximation condition,

$$\begin{aligned} V^*(s) - Q^*(s,\pi(s)) &= V^*(s) - \tilde{Q}(s,\pi(s)) + \tilde{Q}(s,\pi(s)) - Q^*(s,\pi(s)) \\ &\leq V^*(s) - \tilde{Q}(s,\pi^*(s)) + \varepsilon \\ &= Q^*(s,\pi^*(s)) - \tilde{Q}(s,\pi^*(s)) + \varepsilon \\ &\leq 2\varepsilon \end{aligned}$$

Since $V_\pi(s) = Q_\pi(s,\pi(s))$, it follows that

$$\begin{aligned} V^*(s) - V_\pi(s) &= V^*(s) - Q^*(s,\pi(s)) + Q^*(s,\pi(s)) - V_\pi(s) \\ &\leq 2\varepsilon + Q^*(s,\pi(s)) - Q_\pi(s,\pi(s)) \\ &= 2\varepsilon + \gamma E_{s' \sim P(\cdot|s,\pi(s))} \left[V^*(s') - V_\pi(s') \right]. \end{aligned}$$

The result follows from recursing on this equation and using linearity of expectation. □

The following example shows that the previous bound is tight (modified from Bertsekas and Tsitsiklis [1996]).

EXAMPLE 3.1.2. Consider the two state MDP shown in figure 3.1.1. State 1 has two actions, a "stay" self transition and a "go" transition to state 2. State 2 is absorbing. Let the "self" action at state 1 have 0 associated reward and let all other actions have reward $\frac{2\varepsilon}{1-\gamma}$. Clearly the optimal value from all states is $\frac{2\varepsilon}{1-\gamma}$ (recall we use normalized rewards), and the optimal policy chooses "go" at state 1. Consider starting with an optimal policy π. Then $Q_\pi(1,\text{go}) = \frac{2\varepsilon}{1-\gamma}$ and $Q_\pi(1,\text{stay}) = \frac{2\varepsilon\gamma}{1-\gamma}$, and the difference between these state-action values is 2ε. Hence, if we have an approximation error of ε, a greedy update could reverse the preference and set $\pi'(1) = \text{stay}$. For this update, $V_{\pi'}(1) = 0$, so $V_{\pi'}(1) - V_\pi(1) = -\frac{2\varepsilon}{1-\gamma}$ which shows the bound is tight.

The important point to note in this example is how the error compounds. Due to an error at one state, the agent is forced to stay at the state where it has made an error (thus compounding the error at this worst case state in the worst possible manner).

Note that the previous bound doesn't suggest a procedure to approximate Q^* since there is no straightforward means of obtaining samples of Q^* or V^*. The following section addresses the optimality guarantees of approximate iterative dynamic programming schemes which attempt to approximate Q^* or V^*.

3.2. Discounted Approximate Iterative Methods

This section presents results on both discounted approximate value and policy iteration. We start with the approximate value iteration algorithm since it is easier to analyze. However, it should be noted that this algorithm is somewhat more unnatural than the approximate policy iteration algorithm (see Bertsekas and Tsitsiklis [1996] for a discussion of this point).

3.2.1. γ-Discounted Approximate Value Iteration. In the approximate value iteration algorithm, approximate backups of a vector J_t are performed rather than exact backups. Assume that each vector J_t satisfies the following approximation condition

$$||J_t - BJ_{t-1}||_\infty \leq \varepsilon$$

where B is the "backup operator" defined in subsection 2.3.1 and $||x||_\infty = \max_s |x(s)|$. As usual, let $\pi_t(s)$ be the greedy policy

$$\pi_t(s) = \arg\max_a((1-\gamma)r(s,a) + \gamma E_{s' \sim P(\cdot|s,a)} J_t(s'))$$

where for simplicity we have assumed that P is known.

It is too much to hope that such a scheme converges to a single policy. However, the values of the asymptotic set of policies do converge into some region, as the following theorem shows (developed from Bertsekas and Tsitsiklis [1996]).

THEOREM 3.2.1. *Assume the sequence of vectors J_t generated by γ-approximate value iteration satisfies $||J_t - BJ_{t-1}||_\infty \leq \varepsilon$ and that $J_0 = 0$. Then the sequence of greedy policies π_t satisfies*

$$\limsup_{t\to\infty} ||V^* - V_{\pi_t}||_\infty \leq \frac{2\varepsilon}{(1-\gamma)^2}.$$

In addition to the unappealing max norm error, there are two factors of the horizon time $\frac{1}{1-\gamma}$ present in this bound. One might hope for only one factor.

PROOF. Using the approximation condition and the standard contraction property of an exact value iteration update,

$$\begin{aligned} ||V^* - J_t||_\infty &\leq ||V^* - BJ_{t-1}||_\infty + ||BJ_{t-1} - J_t||_\infty \\ &\leq \gamma||V^* - J_{t-1}||_\infty + \varepsilon \end{aligned}$$

Recursing on this equation using $J_0 = 0$,

$$||V^* - J_t||_\infty \leq \gamma^t + \frac{\varepsilon}{1-\gamma}.$$

Hence, as $t \to \infty$, $||V^* - J_t||_\infty \to \frac{\varepsilon}{1-\gamma}$ and the result follows from the greedy update theorem 3.1.1. □

The last equation in the proof along with theorem 3.1.1 imply the following bound on the convergence rate

$$||V^* - V_{\pi_t}||_\infty \leq \frac{2\gamma^t}{1-\gamma} + \frac{2\varepsilon}{(1-\gamma)^2}\,.$$

Note that this bound on the convergence rate of $\frac{2\gamma^t}{1-\gamma}$ is worse than that of exact value iteration which was just $||V^* - V_{\pi_t}||_\infty \leq 2\gamma^t$ (see section 2.3.1).

3.2.2. γ-Discounted Approximate Policy Iteration. In approximate policy iteration, for each update, the value of the policy is approximated with some regression procedure, and then the policy is updated to be greedy with respect to this approximation.

Let π_t be the policy at the t-th step and $\tilde{Q}_t$ be our approximation of $Q_{\pi_t}(s,a)$. The policy at the next timestep is the greedy policy $\pi_{t+1}(s) = \arg\max_a \tilde{Q}_t(s,a)$. Let us assume the following bound on our max norm error at each timestep

$$||\tilde{Q}_t - Q_{\pi_t}||_\infty \leq \varepsilon\,.$$

The following theorem (from Bertsekas and Tsitsiklis [1996]) provides a performance guarantee for this algorithm that is identical to that of approximate value iteration.

THEOREM 3.2.2. *(Bertsekas and Tsitsiklis [1996]) Assume the sequence of approximate state-action values $\tilde{Q}_t$ generated by γ-approximate policy iteration satisfies $||\tilde{Q}_t - Q_{\pi_t}||_\infty \leq \varepsilon$. Then the sequence of policies π_t satisfies*

$$\limsup_{t\to\infty} ||V^* - V_{\pi_t}||_\infty \leq \frac{2\varepsilon}{(1-\gamma)^2}\,.$$

The proof is not provided, since it is somewhat technical. However, the following lemma (developed from Bertsekas and Tsitsiklis [1996]) gives insight into the algorithm. This lemma shows that, even though improvement at each step is not guaranteed, a pseudo contraction property still holds.

LEMMA 3.2.3. *Assume the sequence of approximate state-action values $\tilde{Q}_t$ generated by γ-approximate policy iteration satisfies $||\tilde{Q}_t - Q_{\pi_t}||_\infty \leq \varepsilon$. Then the sequence of policies π_t satisfies*

$$||V^* - V_{\pi_{t+1}}||_\infty \leq \gamma||V^* - V_{\pi_t}||_\infty + \frac{2\varepsilon}{1-\gamma}\,.$$

Note that the theorem immediately follows from this lemma. Also, the lemma implies the following bound on the convergence rate,

$$||V^* - V_{\pi_t}||_\infty \leq \gamma^t + \frac{2\varepsilon}{(1-\gamma)^2}\,.$$

Unlike the case of approximate value iteration, this bound on the convergence rate of approximate policy iteration is identical to its exact counterpart, since for exact policy iteration $||V^* - V_{\pi_t}||_\infty \leq \gamma^t$ (see section 2.3.2) — though of course the limit is different.

Unfortunately, the following example shows this bound is tight (from Bertsekas and Tsitsiklis [1996]) by providing a sequence of policy degradations where the worst case penalty is incurred at every update. Further, the sequence shows that the worst case error could occur at *any single state* and yet still cause maximal performance degradation. This makes the subtle point, that the max norm is the appropriate error to consider for this greedy algorithm (rather than some average error).

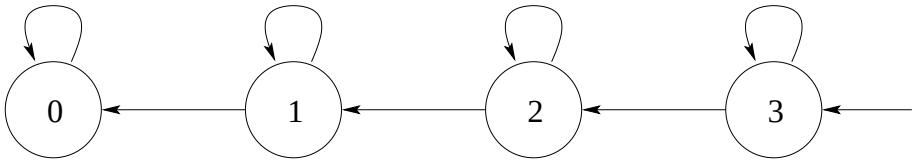

FIGURE 3.2.1. See text for description.

EXAMPLE 3.2.4. Let us consider the infinite state MDP shown in figure 3.2.1. State 0 is absorbing. The other states have two actions. At a state $i \geq 1$, one action is a self transition and the other action transitions to state $i-1$. Let us label these actions as "stay" and "go", respectively. Define the constant ρ as

$$\rho \equiv \frac{2\varepsilon}{(1-\gamma)^2}$$

and define the rewards in terms of ρ as follows:

$$\begin{aligned} r(0) &= \rho \\ r(i, \text{go}) &= \rho \\ r(i, \text{stay}) &= \gamma^i \rho \,. \end{aligned}$$

Clearly the optimal policy is to choose "go" at all states, except for state 0 (which is absorbing), and this policy has value $V^*(i) = \rho$ for all i (again recall we use normalized rewards).

Let us set the initial policy π_0 to be optimal. We now show that at timestep t a the policy π_t could choose to stay at state t, with a value of $V_{\pi_t}(t) = \gamma^t \rho$. This proves the bound is tight since

$$\begin{aligned} V^*(t) - V_{\pi_t}(t) &= \rho - \gamma^t \rho \\ &= \frac{2\varepsilon(1-\gamma^t)}{(1-\gamma)^2} \end{aligned}$$

which approaches $\frac{2\varepsilon}{(1-\gamma)^2}$ for t sufficiently large.

Proceeding inductively, the claim is as follows: at time step t, the policy π_t could stay at the t-th state, and for all states $i > t$, the policy is unaltered (*ie* the go action is chosen). Note that if the difference between the exact state-action values for the go and stay action is 2ε, then due to approximation error, a policy update could choose the stay action.

By assumption, the $t = 0$ base case is true. At time step t, assume the the claim is true. Hence, $V_{\pi_t}(t) = \gamma^t \rho$ which implies

$$\begin{aligned} V_{\pi_t}(t+1) &= (1-\gamma)\rho + \gamma^{t+1}\rho \\ Q_{\pi_t}(t+1, \text{go}) &= V_{\pi_t}(t+1) \\ Q_{\pi_t}(t+1, \text{stay}) &= (1-\gamma) r(t+1, \text{stay}) + \gamma V_{\pi_t}(t+1) \end{aligned}$$

It follows that

$$\begin{aligned} Q_{\pi_t}(t+1, \text{go}) - Q_{\pi_t}(t+1, \text{stay}) &= (1-\gamma)(V_{\pi_t}(t+1) - r(t+1, \text{stay})) \\ &= (1-\gamma)((1-\gamma)\rho + \gamma^{t+1}\rho - \gamma^{t+1}\rho) \\ &= (1-\gamma)^2 \rho \\ &= 2\varepsilon \,. \end{aligned}$$

Hence, the next policy π_{t+1} could choose to stay at state $t+1$. Also, it straight forward to see that the difference between the state-action values for states $i > t+1$ is greater than 2ε, and so the policy update does alter the policy for these states. This proves the claim.

3.3. Approximate Linear Programming

This section highlights the recent results of the approximate linear programming approach of de Farias and Van Roy [2001]. See section 2.3.4 for the exact linear program formulation. In the approximate version, we replace the vector J by the function approximator

$$J_w(s) = \sum_i w_i \phi_i(s)$$

where $\phi_i(s)$ is a feature vector and w_i are the weights. The w_i's are now the variables in the linear program:

$$\begin{aligned} \min_{w_i} \quad & E_{s\sim\mu}\left[\sum_i w_i\phi_i(s)\right] \\ \text{s.t.}\, \forall s,a \quad & \sum_i w_i\phi_i(s) \geq (1-\gamma)r(s,a) + \gamma E_{s'\sim P(\cdot|s,a)}\left[\sum_i w_i\phi_i(s')\right] . \end{aligned}$$

where μ is some probability distribution. The number of these features (and hence the number of weights) can be chosen to be much smaller than the size of the state space.

Let w^* be a solution to this linear program. A bound from de Farias and Van Roy [2001] on the quality of $J_{w^*}(s)$ states that

$$E_{s\sim\mu}|V^*(s) - J_{w^*}(s)| \leq \frac{2}{1-\gamma}\min_w ||V^* - J_w||_\infty .$$

The appealing aspect of this bound is that the quality is stated in terms of some measure of the best approximation possible under the chosen feature set.

Unfortunately, the result is stated in terms on a max norm error. However, de Farias and Van Roy go on to refine this bound in order to take of advantage of the choice of the distribution μ. This weakens the max norm error to a weighted max norm error (see the paper for full details). This result is interesting (and powerful) because it suggests that, with an appropriate choice of the features and weights, we can obtain a good approximation to the optimal value function *on average, ie* our bound on $E_{s\sim\mu}|V^*(s) - J_{w^*}(s)|$ could be small.

Unfortunately, note that the number of constraints is equal to the size of state-action space. However, de Farias and Van Roy [2001] provide a constraint sampling procedure and conditions under which the sample size bounds for obtaining a vector "close" to J_{w^*} is bounded independently of the size of state space and polynomially in the number of features (see de Farias and Van Roy [2001] for details).

Obtaining a bound on the quality of the resulting greedy policy, π_{w^*}, is much trickier. Recall our greedy update bound (theorem 3.1.1) is in terms of the infinity error. Translating a bound on the average error $E_{s\sim\mu}|V^*(s) - J_{w^*}(s)|$ to a bound on the max norm error $||V^*(s) - J_{w^*}(s)||_\infty$ leads to quite a weak result for the quality of π_{w^*}. de Farias and Van Roy present a bound on the quality of π_{w^*} that is stated in terms of which states the π_{w^*} tends to visit. Essentially, the bound states that if π_{w^*} happens to visit states with low approximation error, then this policy is close to optimal.

In general, the problem with greedy updates is that, we can't control which states our greedy policy visits and this leads to the devastating max norm greedy update bound. Though empirically we are often not this unlucky. An important open question is under what formal conditions (or even natural heuristics) do greedy updates provide good policies when we have some average guarantee on J. Conditions on this latter result would be most helpful in understanding the successes (and failures) of greedy value function methods.

CHAPTER 4

Policy Gradient Methods

4.1. Introduction

In recent years, policy gradient methods have seen a rise in popularity as an alternative to approximate value function methods. As discussed in the last chapter, the performance of a greedy policy derived from some approximate value function can be worse than the old policy by an amount related to the *max norm* error. This has motived the use of policy gradient methods which have stronger performance improvement guarantees.

In the policy gradient approach, the goal is to find a good policy among a class of *stochastic policies* parameterized by $\theta \in R^k$ (without recourse to value function methods). The use of stochastic policies for these methods is an interesting (and perhaps questionable) option since every MDP has a deterministic optimal policy (and there is a trivial transformation to go from any stochastic optimal policy to a deterministic optimal policy). However, with an impoverished policy class or partial information, stochastic policies can be useful (see Singh, Jaakkola, and Jordan [1994]).

Policy gradient methods attempt to adjust the parameters in the direction of the gradient of the performance measure. For large scale problems or when the transition model is unknown, the gradient is not efficiently or exactly computable, and simulation methods are typically used to estimate the gradient. In this chapter, the sample complexity issues related to simulation based policy gradient methods are examined. The most striking problem is that gradient methods intertwine exploration and exploitation, which could lead to an unreasonably large sample complexity.

4.1.1. Background. Simulation based gradient algorithms to optimize the average reward performance criterion have a long history in a variety of related fields (see Baxter and Bartlett [2001] for a review of these likelihood ratio methods). Gradient algorithms for Markov decision processes were provided by Glynn [1986] and Williams [1992]. Here, the gradient is estimated using sequences of states and rewards encountered between visits to some designated recurrent state, and the parameter θ is updated during every recurrence cycle, *ie* between every visit to the recurrent state. Eligibility traces can be used to efficiently perform these updates in an online manner (see Marbach and Tsitsiklis [2001]).

One problematic issue is that the variance in the gradient estimate grows with the recurrence time. This time can often be unreasonably large in large-scale problems. This time is also dependent on the policy, so as performance improves, it is possible that the recurrence time can increase.

A number of more recent approaches present variants to deal with the case when the recurrence time is excessive (Kimura, Yamamura, and Kobayashi [1995]; Marbach and Tsitsiklis [2001]; Baxter and Bartlett [2001]). These approaches introduce a discount factor in

order to obtain biased estimates of the gradient with lower variance. The idea is that the "mixing time" determines the effective recurrence time, and, by using a discount factor, the gradient can be estimated over this effective recurrence time (Baxter and Bartlett [2001]). Informally, the "mixing time" is the time until the stationary distribution is reached.

An additional appealing aspect of policy based gradient methods is their applicability to POMDPs (Baird and Moore [1999]). Intuitively, the underlying reason why gradient methods are applicable to POMDPs is that one can consider the restricted class of policies to be a class which only uses the observable data from the underlying MDP. Some approaches attempt to exploit gradient methods for memory purposes in POMDPs (Peshkin, Meuleau, Kim, and Kaelbling [1999]).

Though standard policy based gradient approaches don't involve value function approximation, a few papers have addressed the connections between gradient methods and actor-critic methods (Barto, Sutton, and Anderson [1983]). Baird and Moore [1999] present an approach which combines value function methods with policy search methods through the choice of the performance measure. Sutton, McAllester, Singh, and Mansour [2000] and Konda and Tsitsiklis [2000] examine approaches where function approximators can be used in lieu of empirical estimates of the state-action values. Kakade [2002] points out that these later approaches have strong connections to a natural (covariant) gradient method (as in Amari [1998]).

4.1.2. The Question of Sample Complexity. Perhaps the most important question for gradient methods is "*How many samples are required before a policy gradient method finds a good policy, in the sense that the this policy can be compared favorably within our restricted class of policies (or some sensible subset)?*". Unfortunately, there are few results in the literature that shed light on the answer to this question.

Marbach and Tsitsiklis [2001] present asymptotic convergence results that prove stochastic gradient ascent algorithms asymptotically reach a point where the gradient is zero. Though important from a consistency point of view, this does us not help us answer our aforementioned question. Also, in the limit of an infinite amount of data, it is reasonable to desire an algorithm which finds an optimal policy (at least if the MDP is finite).

Closer in spirit to our question, Bartlett and Baxter [2000] examine the sample size sufficient for obtaining an accurate gradient estimate. More specifically, they examine the number of samples sufficient in order for each component of the gradient to be ε-close (in magnitude) to its true value. This analysis stresses the importance of the mixing time of the process in obtaining accurate gradient estimates. However, at least one crucial question is left unanswered, and that is when the the gradient is "small", *ie* when it is difficult to obtain accurate estimates of the gradient, what is the quality of our policy?

This chapter examines the aforementioned question of sample complexity. It is argued that policy gradient methods can require an unreasonably large number of samples before a good policy is obtained. Essentially, the lack of exploration in these methods leads to an unreasonably (and an arbitrarily) large sample complexity in order to obtain an accurate estimate of the gradient direction. The problem is that due to a lack of exploration the gradient *magnitude* could be arbitrarily small making it difficult to estimate the gradient *direction* accurately, which is the quantity of interest. Furthermore, a small gradient magnitude does not necessarily imply that the policy is close to any local (or global) optima. It is also argued that the "mixing time" is a red herring in this variance problem caused by the lack of exploration.

4.2. Sample Complexity of Estimation

This section analyzes the sample size sufficient to obtain an accurate gradient estimate (in magnitude). For clarity, this analysis focuses on the T-step setting, where the sampling procedures are simpler since unbiased estimates can be obtained easily (in contrast to the discounted setting, where a cutoff time is often imposed).

4.2.1. Future Distributions. This subsection defines the "future distributions" of a policy π. These distributions are useful throughout this chapter and all of part 2.

We start with the T-epoch case. In the definition, $\Pr(s_t = s|\pi, s_0, M)$ is the probability that the state at time t is s when the policy π is followed in M starting from state s_0 (see section 2.1 for a definition of this induced stochastic process by π on M).

DEFINITION 4.2.1. Let M be a T-epoch MDP with state space $\mathcal{S}$, π be a policy with respect to M, and s_0 be a starting state for M. The **future state-time distribution** $d_{\pi,s_0,M}(s,t)$ on $\mathcal{S} \times \{0,1,\ldots T-1\}$ is

$$d_{\pi,s_0,M}(s,t) \equiv \frac{1}{T}\Pr(s_t = s|\pi, s_0, M)\,.$$

When clear from context, we suppress the M dependence of $d_{\pi,s_0,M}$. Note that the distribution is properly normalized and has a simple sampling procedure. To sample from d_{π,s_0}, first uniformly choose a time $t \in \{0,\ldots,T-1\}$ and then choose a state s according to $\Pr(s_t = s|\pi, s_0, M)$. The sample (s,t) is distributed according to d_{π,s_0}.

The definition for the γ-discounted distribution follows (see Sutton, McAllester, Singh and Mansour [2000] for a similar definition). We only define this distribution for stationary policies.

DEFINITION 4.2.2. Let M be an infinite horizon MDP with state space $\mathcal{S}$, π be a stationary policy with respect to M, γ be a discount factor and s_0 be a starting state for M. The **γ-discounted future state distribution** $d_{\pi,s_0,\gamma,M}(s)$ on $\mathcal{S}$ is

$$d_{\pi,s_0,\gamma,M}(s) \equiv (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \Pr(s_t = s|\pi, s_0, M)\,.$$

Again, this distribution is properly normalized. Unlike for the T-epoch case, this distribution is just over states (which is motivated by the fact that π is stationary). A sampling procedure is also straightforward. Start the MDP in state s_0, and simulate π. Accept each state as the sample with probability $1-\gamma$. The accepted state is then distributed according to $d_{\pi,s_0,\gamma}$.

Note that we can write the value functions in terms of these distributions. For a T-epoch MDP,

$$V_\pi(s_0) = E_{(s,t)\sim d_{\pi,s_0}} E_{a\sim\pi(\cdot|s,t)}\left[r(s,a)\right]$$

and for an infinite horizon MDP where π is stationary

$$V_{\pi,\gamma}(s_0) = E_{s\sim d_{\pi,s_0,\gamma}} E_{a\sim\pi(\cdot|s)}\left[r(s,a)\right]\,.$$

These relations suggests that these future distributions are natural to consider.

4.2.2. The Policy Gradient. For the T-epoch case we consider non-stationary policies of the form $\pi(a|s,t,\theta)$, and for the γ-discounted case, we consider stationary policies of the form $\pi(a|s,\theta)$. We also assume that the derivatives $\nabla\pi$ exist.

THEOREM 4.2.3. *Let M be a T-epoch MDP and let $\pi(a|s,t,\theta)$ be a parameterized policy. Then*

$$\nabla V_\pi(s_0) = T E_{(s,t)\sim d_{\pi,s_0}}\left[\sum_a \nabla\pi(a|s,t,\theta) Q_{\pi,t}(s,a)\right]$$

Note that the natural form of this gradient is in terms of an *expectation* over the state-times but a *sum* over actions. This has important implications for sampling methods.

PROOF. For notational convenience, we use $P_t(s)$ for $\Pr(s_t = s|\pi, s_0, M)$. Hence,

$$\begin{aligned}
& E_{s\sim P_t}\left[\nabla V_{\pi,t}(s)\right] \\
= \;& E_{s\sim P_t}\left[\nabla E_{a\sim\pi(\cdot|s,t,\theta)}\left[Q_{\pi,t}(s,a)\right]\right] \\
= \;& E_{s\sim P_t}\left[\sum_a \nabla\pi(a|s,t,\theta) Q_{\pi,t}(s,a)\right] + E_{s\sim P_t} E_{a\sim\pi(\cdot|s,t,\theta)}\left[\nabla Q_{\pi,t}(s,a)\right] \\
= \;& E_{s\sim P_t}\left[\sum_a \nabla\pi(a|s,t,\theta) Q_{\pi,t}(s,a)\right] + \\
& E_{s\sim P_t} E_{a\sim\pi(\cdot|s,t,\theta)}\left[\nabla(\frac{1}{T} r(s,a) + E_{s'\sim P(\cdot|s,a)} V_{\pi,t+1}(s'))\right] \\
= \;& E_{s\sim P_t}\left[\sum_a \nabla\pi(a|s,t,\theta) Q_{\pi,t}(s,a)\right] + E_{s\sim P_{t+1}}\left[\nabla V_{\pi,t+1}(s)\right]
\end{aligned}$$

where the last step uses the definition of P_t. Since $\nabla V_\pi(s_0) = E_{s\sim P_0}\left[\nabla V_{\pi,0}(s)\right]$, recursing on the previous equation and using $V_{\pi,T} = 0$ leads to

$$\begin{aligned}
\nabla V_\pi(s_0) &= \sum_{t=0}^{T-1} E_{s\sim P_t}\left[\sum_a \nabla\pi(a|s,t,\theta) Q_{\pi,t}(s,a)\right] \\
&= T E_{(s,t)\sim d_{\pi,s_0}}\left[\sum_a \nabla\pi(a|s,t,\theta) Q_{\pi,t}(s,a)\right]
\end{aligned}$$

where the last step uses the definition of d_{π,s_0}. □

The policy gradient for the discounted case is as follows.

THEOREM 4.2.4. *Let M be an infinite horizon MDP and let $\pi(a|s,\theta)$ be a stationary parameterized policy. Then*

$$\nabla V_{\pi,\gamma}(s_0) = \frac{1}{1-\gamma} E_{s\sim d_{\pi,s_0,\gamma}}\left[\sum_a \nabla\pi(a|s,\theta) Q_{\pi,\gamma}(s,a)\right]$$

The proof for the discounted case is from Sutton, McAllester, Singh, and Mansour [2000] and is included for completeness.

PROOF. Again, for notational convenience, we use $P_t(s)$ for $P(s_t = s|\pi, s_0, M)$.

$$\begin{aligned}
& E_{s\sim P_t}\left[\nabla V_{\pi,\gamma}(s)\right] \\
= \; & E_{s\sim P_t}\left[\sum_a \nabla\pi(a|s,\theta)Q_{\pi,\gamma}(s,a)\right] + E_{s\sim P_t}E_{a\sim\pi(\cdot|s,\theta)}\left[\nabla Q_{\pi,\gamma}(s,a)\right] \\
= \; & E_{s\sim P_t}\left[\sum_a \nabla\pi(a|s,\theta)Q_{\pi,\gamma}(s,a)\right] + \\
& E_{s\sim P_t}E_{a\sim\pi(\cdot|s,t,\theta)}\left[\nabla((1-\gamma)r(s,a) + \gamma E_{s'\sim P(\cdot|s,a)}V_{\pi,\gamma}(s'))\right] \\
= \; & E_{s\sim P_t}\left[\sum_a \nabla\pi(a|s,\theta)Q_{\pi,\gamma}(s,a)\right] + \gamma E_{s\sim P_{t+1}}\left[\nabla V_{\pi,\gamma}(s)\right]
\end{aligned}$$

where the last step uses the definition of P_t. Since $\nabla V_{\pi,\gamma}(s_0) = E_{s\sim P_0}\left[\nabla V_{\pi,\gamma}(s)\right]$, recursing on the previous equation leads to

$$\begin{aligned}
\nabla V_{\pi,\gamma}(s_0) &= \sum_{t=0}^{\infty}\gamma^t E_{s\sim P_t}\left[\sum_a \nabla\pi(a|s,\theta)Q_{\pi,\gamma}(s,a)\right] \\
&= \frac{1}{1-\gamma}E_{s\sim d_{\pi,s_0,\gamma}}\left[\sum_a \nabla\pi(a|s,\theta)Q_{\pi,\gamma}(s,a)\right]
\end{aligned}$$

where we have used the definition of $d_{\pi,s_0,\gamma}$. □

4.2.3. Estimation. We now analyze the number of samples sufficient to obtain an accurate gradient estimate in magnitude. An important point is that the form of the gradient formula is in terms of an expectation over the state space and so sampling methods can be applied for estimation (with no dependence on the size of the state space).

First, note the presence of a *sum* over actions, rather than an expectation. The standard way to deal with this is to write the gradient as:

$$\nabla V_\pi(s_0) = TE_{(s,t)\sim d_{\pi,s_0}}E_{a\sim\pi(\cdot|s,t,\theta)}\left[\frac{\nabla\pi(a|s,t,\theta)}{\pi(a|s,t,\theta)}Q_{\pi,t}(s,a)\right]$$

which suggests an "on-policy" importance sampling procedure, where one follows the policy π and estimates the gradient from sample trajectories from the start state s_0.

Unfortunately, this procedure is unappealing when π is close to a deterministic policy, since then the factor of $\frac{\nabla\pi(a|s,t,\theta)}{\pi(a|s,t,\theta)}$ could become arbitrarily large. One could assume that $\frac{\nabla\pi}{\pi}$ is bounded by a constant over all s, a, θ, but this is a rather unnatural assumption for some near deterministic policies (since this constant must be chosen to be quite large).

Instead, let us explicitly recognize the lack of an expectation over the actions and treat estimation as an importance sampling problem. Ideally, we would like to incorporate domain knowledge in the choice of an importance sampling distribution to decrease the variance, but without this knowledge, let us use the uniform distribution over the action space. We write the gradient as:

$$\nabla V_\pi(s_0) = ATE_{(s,t)\sim d_{\pi,s_0}}E_{a\sim\text{Uniform}}\left[\nabla\pi(a|s,t,\theta)Q_{\pi,t}(s,a)\right] .$$

where A is the size of the action space. This form implies a sampling procedure with the uniform distribution.

A procedure to obtain an unbiased estimate of the gradient is as follows. First, we obtain a sample $(s,t) \sim d_{\pi,s_0}$ (as described in section 4.2.1) and an $a \sim$ Uniform. Then obtain an unbiased estimate $\hat{Q}$ of $Q_{\pi,t}(s,a)$ by simulating π for the remaining $T-t$ steps and using the empirical return to construct $\hat{Q}$. An unbiased estimate of the gradient is $AT\nabla\pi(a|s,t,\theta)\hat{Q}$. This procedure requires T samples. It should be clear that the only modelling requirements for this procedure are the need to "reset" to s_0 and the ability to act in the MDP.

Algorithm 5 EstimateGradient(θ, m)

(1) Obtain m samples of the form $\{a_i, s_i, t_i, \hat{Q}_i)\}$ where
   (a) $s_i, t_i \sim d_{\pi,s_0}$ and $a_i \sim$ Uniform
   (b) $\hat{Q}_i$ is an estimate of $Q_{\pi,t_i}(s_i,a_i)$

(2) Set
$$\widehat{\nabla V} = \frac{AT}{m}\sum_i \nabla\pi(a_i|s_i,t_i,\theta)\hat{Q}_i$$

(3) Return $\widehat{\nabla V}$

The EstimateGradient algorithm is shown in algorithm 5. This algorithm constructs m estimates, as described above, and returns the average of these m estimates. The number of observed samples by this algorithm is mT. The following theorem provides a bound on the sample complexity for obtaining an accurate (in magnitude) estimate of the gradient. Here, ∇_k is the k-th component of the gradient.

THEOREM 4.2.5. *(Gradient Sample Size) Assume that $\theta \in R^k$ and $\nabla\pi \leq B$. With an appropriate choice of m and upon input of a parameter θ and m, EstimateGradient observes*

$$O(\frac{A^2T^3B^2}{\varepsilon^2}\log\frac{k}{\delta})$$

transitions and with probability greater than $1-\delta$, the output $\widehat{\nabla_k V}$ satisfies, for all k,

$$|\widehat{\nabla_k V} - \nabla_k V_\pi(s_0)| \leq \varepsilon\,.$$

PROOF. Each estimate is bounded in the interval $[0, ATB]$. By Hoeffding's and the union bound, the probability that there exists an i such that our estimate of $\widehat{\nabla_i V}$ is ε inaccurate is less than $k\exp(\frac{-2B^2A^2T^2m}{\varepsilon^2})$. Hence, if

$$m = \frac{B^2A^2T^2}{\varepsilon^2}\log\frac{k}{\delta}$$

then this probability is less than δ. The total number of observed transitions is $O(mT)$. □

The immediate question is what is the quality of the policy when we can no longer obtain an accurate gradient estimate. The next section argues that due to the lack of exploration this policy can be arbitrarily poor.

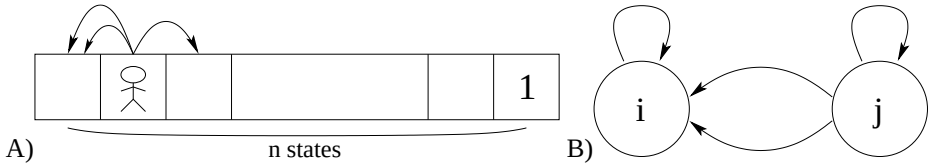

FIGURE 4.3.1. A) MDP where two actions move agent to the left and one actions moves agent to the right. B) A two state MDP.

4.3. The Variance Trap

Essentially, the lack of exploration in gradient methods translates into arbitrarily large variance in obtaining an accurate *direction* of the gradient. The previous analysis only guarantees that the gradient is accurate in *magnitude*. If the gradient is small in magnitude, then we don't have a guarantee on obtaining an accurate direction, which is what is necessary for policy improvement.

Consider the MDP shown in Figure 4.3.1A (adapted from Thrun [1992]), where the agent starts in the leftmost state. Two actions take the agent to the right and one action takes the agent to the left. There are 50 states, and let us set $T = 50$. Obviously, the optimal policy always chooses the right most action and reaches the goal state in the horizon time. Now let us consider the time to the goal under a policy that gives equal probability to all actions. Under this policy, we have an equal probability of any T-step sequence of actions $a_1, a_2, \ldots a_T$, and only one such sequence reaches the goal. There are 3^{50} such sequences and so the probability of reaching the goal state in T steps is roughly 10^{-25}. Thus, any "on-policy" method using this random walk policy has to run for about 10^{25} in order to just reach the reward state. Instead if $T = 60$, then the chance of reaching the goal in T-timesteps under a random walk policy is roughly 10^{-18}.

This MDP falls into the class of MDPs in which random actions are more likely than not to increase the distance to the goal state. For these classes of problems (see Whitehead [1991]), the expected time to reach the goal state using undirected exploration, *ie* random walk exploration, is exponential in the size of the state space. It is clear that a large class of problems fall into this category.

Now let us return to gradient methods. Typically, the parameter θ is initialized randomly, which often leads to random policies. Hence, for this problem, any sensible estimate of the gradient without reaching the goal state would be zero and obtaining a non-zero gradient estimate with "on-policy" samples has an *exponential* dependence on our horizon time (eventhough there are only 50 states).

In this example, note that a zero estimate is a rather accurate estimate of the gradient in terms of *magnitude*, but this provides no information about the *direction*, which is the crucial quantity of interest in order to improve the policy. The analysis above (and in Bartlett and Baxter [2000]) shows that the magnitude of the gradient can be estimated accurately (up to ε tolerance). However, this only implies a a correct direction if the magnitude is larger than ε. As this example shows, the magnitude of the gradient can be very small *even when the performance of the policy is not close to optimal*. Furthermore, note that the random walk policy "mixes" quickly in this example, which shows that the mixing time results in Baxter and Bartlett [2002] are not a relevant factor for this problem.

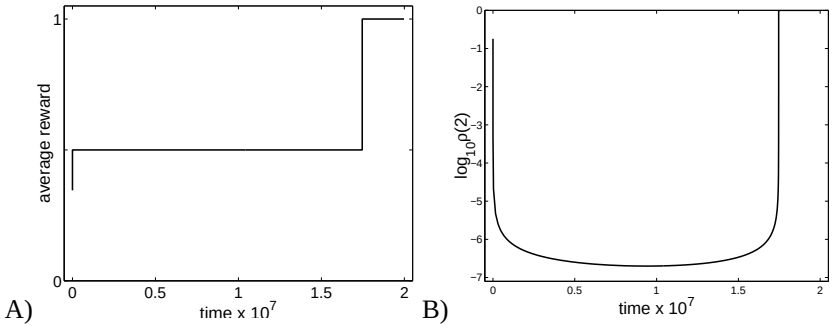

FIGURE 4.3.2. A)The average reward vs. time (on a 10^7 scale) of a policy under standard gradient descent in the limit of an infinitesimally small learning rate (initial conditions stated in text). B) The stationary probability of state j vs. time (on a 10^7 scale).

Importance sampling methods (Precup, Sutton, and Dasgupta [2001]; Meuleau, Peshkin, and Kim [2001]) have been considered in reinforcement learning and provide "off-policy" methods. Loosely, in an off-policy method, the agent can act according to any policy, and importance weights are used for appropriate corrections. Unfortunately, these do not provide feasible solutions for this class of problems. The reason is that if the agent could follow some "off-policy" trajectory to reach the goal state in a reasonable amount of time, the importance weights would have to be exponentially large.

The following additional example demonstrates that estimating a gradient could become arbitrarily difficult for a simple two state MDP. Consider the MDP shown in Figure 4.3.1B, where each state has a self transition action and a cross transition action. The cross transition actions are not rewarding and the self-transition has a reward of $.5$ at state i and a reward of 1 at state j. It is clear that the optimal policy just stays at state j for maximal reward.

For simplicity, we consider the average reward setting for this problem (which has effectively the same behavior as that of the "large" T case). Now let us consider some initial policy which has the stationary distribution $\rho(i) = .8$ and $\rho(j) = .2$. Under this policy, the self-transition at action at i looks rewarding since $Q_\pi(i, \text{self}) > Q_\pi(i, \text{cross})$. Note that if the probability of the self transition action at state i is increased, then this decreases the probability of visiting state j. However, it is state j where learning must occur at, if the agent is to act near-optimally. In fact, as long as the the agent does not improve the policy at state j, the self transition at i looks preferable.

Recall that the gradient weights the contribution from a particular state by its future state distribution. Hence, the higher state visitation frequency at state i might have a self-reinforcing effect — the more the agent visits state i the more the agent will reinforce the self transition at state i (and perhaps suppress the learning to state j where ultimately learning must occur, if the agent is to stop reinforcing state i). The question is then can the agent get trapped reinforcing only the self transition at state i, due to a lack of exploration.

Let us use the common Gibbs table-lookup distribution, $\{\pi_\theta : \pi(a|s) \propto \exp(\theta_{sa})\}$. This parameterization is interesting because it is capable of representing *all* policies (except those completely deterministic policies which can be approximated to arbitrary precision in this class). Under an initial policy that has the stationary distribution $\rho(i) = .8$ and

$\rho(j) = .2$ (using $\pi(\text{stay}|i) = .8$ and $\pi(\text{stay}|j) = .9$), the agent reinforces the action at state i more than that of state j, which has the effect of decreasing the probability of visiting state j. This leads to an an extremely flat plateau of improvement at $.5$ average reward shown in Figure 4.3.2A. Figure 4.3.2B shows that this problem is so severe that $\rho(j)$ drops as low as 10^{-7} from its initial probability of $.2$. Again, in this example, the policy "mixes" extremely quickly.

These results suggest that in any reasonable number of steps, a gradient method could end up being trapped at plateaus where estimating the gradient direction has an unreasonably large sample complexity, yet the performance of the policy is not even near to any local optima.

Part 2

Sample Based Planning

CHAPTER 5

The "Mismeasure" of Reinforcement Learning

In the previous chapter, we saw that the policy improvement of a parameterized policy π is inherently linked to *expectations* with respect to the future state distribution of π. The intuitive reason as to why the future state distribution of π is relevant is because when dealing with a restricted class of policies, the policy gradient must take into account the states where π visits frequently in order to avoid making errors at these states, since these errors are more costly and could lead to policy degradation. However, in order to find a near-optimal policy it is often the case that policy improvement must occur at states which are not visited often under the current policy. This latter issue is related to the problem of exploration and leads to policy gradient methods having excessive sample complexity.

In contrast, exact dynamic programming methods uniformly improve the policy at all states, which obviates the need for exploration. Furthermore, most performance bounds are stated in terms of a max norm error, which doesn't stress the importance of any particular set of states. Bertsekas [1987] and Singh and Yee [1994] provide a performance bound that depends on the maximum difference between a vector J and the optimal value function V^*, *ie* $||J - V^*||_\infty$ (see theorem 3.1.1). An alternative bound of Williams and Baird [1993] is presented in terms of the Bellman error of a vector J, which does not depend on V^*. The Bellman error is the maximum difference between J and the one-step lookahead of J, *ie* $||BJ - J||_\infty$ (where B is the backup operator defined in subsection 2.3.1). Most sensible value function methods attempt to minimize this Bellman error. Unfortunately, these max norm errors are not amenable to sampling-based methods, unlike the situation for gradient methods where expectations with respect to the future state distributions can be easily evaluated.

This chapter provides bounds and algorithms in which max norm statements can be avoided. Since it is often the policy that we ultimately care about, we do not wish to be tied to using only value function methods and desire bounds that are more generally applicable. Instead of stating bounds in terms of a vector J, the difference between the performance of a policy π and that of optimal is stated in terms of the *advantages* of the policy π (as defined in Baird [1993]) and in terms of an *expectation* with respect to the future state distribution of an optimal policy. Informally, the advantage of a policy at state action (s, a) is the amount by which the performance will increase by taking action a in state s. Rather intuitively, this result shows that if we desire to find a policy π which competes favorably against some optimal policy π^* then we desire an algorithm which minimizes the advantages of π at the states where π^* tends to visit. Essentially, if π has no large advantages at states where π^* tends to visit then there is not much room to improve π.

Additionally, this result directly motivates a *non-stationary* approximate policy iteration algorithm (NAPI), which is a generalization of undiscounted value iteration (see subsection 2.3.1). This algorithm assumes access to a "black box" PolicyChooser algorithm, which

outputs greedy policies that are used by NAPI. This algorithm enjoys a performance bound in which the performance difference between the output policy and that of an optimal policy is dependent only on an *average error* of the PolicyChooser. Intuitively, this average is taken with respect to the future state distribution of an optimal policy.

This chapter also briefly examines implementing the PolicyChooser using a regression algorithm and compares the results with the function approximation methods of chapter 3. In this setting, the discrepancy between the performance of the policy returned by NAPI and that of optimal is just an *average regression error.* Here, we do not explicitly consider the sample complexity of this implementation (though much work has gone into understanding the sample complexity of regression algorithms with respect to an average error, see Anthony and Bartlett [1999] for review). In the next chapter, we consider a more natural policy search setting, where sample size bounds are provided.

The bounds and algorithms presented here suggests interesting connections to the supervised learning setting. In the supervised learning setting, we obtain a "training set" of the form $\{(x, y)\}$ from a distribution $P(x, y)$ and our test error is probed by examining our performance on a test set $\{(x, ?)\}$ which is sampled according to the input distribution $P(x)$. In contrast to the standard supervised learning setting, where we typically "train" and "test" under the same distribution $P(x)$, the results in this chapter show how the reinforcement learning problem can be viewed as a supervised learning problem where we are "tested" under a (possibly) unknown input measure $Q(x)$ (where $Q(x)$ turns out to be the future state distribution of an optimal policy). This is referred to as the "mismeasure" of reinforcement learning.

5.1. Advantages and the Bellman Error

First, let us define the advantages of a policy (similar to Baird [1993]).

DEFINITION 5.1.1. Let M be a T-epoch MDP and π be a policy with respect to M. The **t-step undiscounted advantage** $A_{\pi,t}(s, a)$ is

$$A_{\pi,t,M}(s, a) = Q_{\pi,t,M}(s, a) - V_{\pi,t,M}(s) .$$

Let M be an infinite horizon MDP, π be a stationary policy with respect to M, and γ be a discount factor. The **γ-discounted advantage** $A_{\pi,\gamma}(s, a)$ is

$$A_{\pi,\gamma,M}(s, a) = Q_{\pi,\gamma,M}(s, a) - V_{\pi,\gamma,M}(s) .$$

It is clear that the advantages are bounded between $[-1, 1]$. The advantage $A_{\pi,t}(s, a)$ is the amount by which the t-value at state s, $V_{\pi,t}(s)$, increases if action a is taken at time t instead of following $\pi(\cdot|s, t)$. The interpretation is analogous for the discounted case.

Another useful quantity is the Bellman error (or Bellman residual), which is a max-norm error over the state-action space. It is typically defined with respect to some vector J in the state space and we only define it for the discounted case.

DEFINITION 5.1.2. Let M be an MDP and let J be a vector on the state space. Define $Q_J(s, a) \equiv (1 - \gamma)r(s, a) + \gamma E_{s' \sim P(\cdot|s,a)} J(s')$. The **Bellman error** B_J with respect to J is:

$$B_J(s) \equiv \sup_{s,a} |Q_J(s, a) - J(s)| .$$

Note the similarity between the Bellman error and the advantages. If J is the value of a policy π, *ie* $J = V_{\pi,\gamma}$, then B_J is just the maximal advantage $||A_{\pi,\gamma}||_\infty$. The following theorem restates the error bound from Williams and Baird [1993] for a greedy policy π based on J (*ie* $\pi = \arg\max_a Q_J(s,a)$).

THEOREM 5.1.3. *(Williams and Baird [1993]) Let M be an MDP and let J be a vector on the state space of M with a Bellman error B_J. The greedy policy π with respect to J satisfies for all states s*

$$V_{\pi,\gamma}(s) \geq V^*_\gamma(s) - \frac{2B_J}{1-\gamma}\,.$$

Exact dynamic programming algorithms minimize this Bellman error (see section 2.3). However, minimizing this Bellman error is much trickier in an approximate setting.

5.2. Performance Differences

The following lemma shows how the difference between two policies can be stated in terms of the advantages of one policy and the future state distribution of the other policy (recall the definition of the future distributions from subsection 4.2.1). This result is *not* stated in terms of max norms, but instead is stated in terms of expectations with respect to future state distributions. This key lemma is useful throughout this chapter and the next two chapters.

LEMMA 5.2.1. *(Performance difference) Let M be an MDP.*

(Undiscounted) If T is finite, then for all policies π and π', and for all s_0,

$$V_{\pi'}(s_0) - V_\pi(s_0) = T E_{s,t\sim d_{\pi',s_0}} E_{a\sim\pi'(\cdot|s,t)}\left[A_{\pi,t}(s,a)\right]$$

(Discounted) If T is infinite, then for all stationary policies π and π', and for all s_0 and γ,

$$V_{\pi',\gamma}(s_0) - V_{\pi,\gamma}(s_0) = \frac{1}{1-\gamma} E_{s\sim d_{\pi',s_0,\gamma}} E_{a\sim\pi'(\cdot|s)}\left[A_{\pi,\gamma}(s,a)\right]$$

Importantly, this result is stated in terms of an *arbitrary* policy π' rather than an optimal one. The lemma shows that π competes favorably against π' if the advantages of π are small where π' tends to visit. Thus policy π is near-optimal if the advantages $A_{\pi,\gamma}$ are small at the state-actions that are frequently visited by some optimal (or near-optimal) policy. This suggests that we desire an algorithm which is capable of minimizing the advantages under the future distribution of a good policy.

Before proving this lemma, a high level outline of the proof is provided. If the agent is in state s and deviates from π at only the first timestep by choosing an action from π', then the change in performance is $E_{a\sim\pi'(\cdot|s,0)}\left[A_{\pi,0}(s,a)\right]$. To switch from policy π to π', one can consider deviating from π to π' at every step. The contribution to the performance change at the t-th step when the state is $s_t = s$ is then $E_{a\sim\pi'(\cdot|s,t)}\left[A_{\pi,t}(s,a)\right]$. We must then sum these performance changes over the horizon and take an appropriate expectation over the states. This leads to the expectation with respect to d_{π',s_0}.

The statement for the infinite horizon setting was proved in Kakade and Langford [2002]. For completeness, the proof is included.

PROOF. The proof for the finite T-case is as follows. As in section 2.1, $\Pr(\cdot|\pi', s_0, M)$ is the distribution over T length paths $(\vec{s}, \vec{a})$ under the policy π' starting from s_0 in M, where $\vec{s}$ is the sequence $(s_0, \ldots s_{T-1})$ and $\vec{a}$ is the sequence $(a_0, \ldots a_{T-1})$. For notational convenience, we write this distribution $\Pr(\cdot|\pi', s_0, M)$ as $P_{\pi'}$ (and explicitly maintain the π' dependence). By definition of the value functions,

$$\begin{aligned}
& V_{\pi'}(s_0) - V_{\pi}(s_0) \\
= \; & \frac{1}{T} E_{\vec{s},\vec{a}\sim P_{\pi'}} \left[\sum_{t=0}^{T-1} r(s_t, a_t) \right] - V_{\pi}(s_0) \\
= \; & E_{\vec{s},\vec{a}\sim P_{\pi'}} \left[\sum_{t=0}^{T-1} \frac{1}{T} r(s_t, a_t) + V_{\pi,t}(s_t) - V_{\pi,t}(s_t) \right] - V_{\pi}(s_0) \\
= \; & E_{\vec{s},\vec{a}\sim P_{\pi'}} \left[\sum_{t=0}^{T-1} \frac{1}{T} r(s_t, a_t) + V_{\pi,t+1}(s_{t+1}) - V_{\pi,t}(s_t) \right]
\end{aligned}$$

where we have rearranged the sum and used the fact that $V_{\pi,0}(s_0) = V_{\pi}(s_0)$ and $V_{\pi,T}(s) = 0$. Define Uniform to be the uniform distribution on $\{0, 1, \ldots, T-1\}$. Since $s_{t+1} \sim P(\cdot|s_t, a_t)$, we can use the definition of $Q_{\pi,t}$ to write the last term as:

$$\begin{aligned}
= \; & E_{\vec{s},\vec{a}\sim P_{\pi'}} \left[\sum_{t=0}^{T-1} Q_{\pi,t}(s_t, a_t) - V_{\pi,t}(s_t) \right] \\
= \; & E_{\vec{s},\vec{a}\sim P_{\pi'}} \left[\sum_{t=0}^{T-1} A_{\pi,t}(s_t, a_t) \right] \\
= \; & T E_{\vec{s},\vec{a}\sim P_{\pi'}} E_{t\sim \text{Uniform}} \left[A_{\pi,t}(s_t, a_t) \right] \\
= \; & T E_{t\sim \text{Uniform}} E_{s\sim \Pr(s_t = s|\pi', M, s_0)} E_{a\sim\pi'(\cdot|s,t)} \left[A_{\pi,t}(s, a) \right] \\
= \; & T E_{s,t\sim d_{\pi', s_0}} E_{a\sim\pi'(\cdot|s,t)} \left[A_{\pi,t}(s, a) \right] .
\end{aligned}$$

The last step follows from the definition of d_{π', s_0}.

The proof for the infinite horizon statement is analogous to the previous proof. Let $P_{\pi'}$ be the distribution $\Pr(\cdot|\pi', M, s_0)$, *ie* $P_{\pi'}$ *is* over infinite length sequences $(\vec{s}, \vec{a})$ which are generated under the policy π' starting from s_0 in M, and where $\vec{s} = (s_0, s_1, \ldots)$ and $\vec{a} = (a_0, a_1 \ldots)$.

$$\begin{aligned}
& V_{\pi',\gamma}(s_0) - V_{\pi,\gamma}(s_0) \\
= \; & (1-\gamma) E_{\vec{s},\vec{a}\sim P_{\pi'}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] - V_{\pi,\gamma}(s_0) \\
= \; & E_{\vec{s},\vec{a}\sim P_{\pi'}} \left[\sum_{t=0}^{\infty} \gamma^t ((1-\gamma) r(s_t, a_t) + V_{\pi,\gamma}(s_t) - V_{\pi,\gamma}(s_t)) \right] - V_{\pi,\gamma}(s_0) \\
= \; & E_{\vec{s},\vec{a}\sim P_{\pi'}} \left[\sum_{t=0}^{\infty} \gamma^t ((1-\gamma) r(s_t, a_t) + \gamma V_{\pi,\gamma}(s_{t+1}) - V_{\pi,\gamma}(s_t)) \right]
\end{aligned}$$

where we have rearranged the sum. Since $s_{t+1} \sim P(\cdot|s_t, a_t)$, we can use the definition of $Q_{\pi,\gamma}$ to write the last term as:

$$\begin{aligned}
&= E_{\vec{s},\vec{a}\sim P_{\pi'}}\left[\sum_{t=0}^{\infty}\gamma^t(Q_{\pi,\gamma}(s_t,a_t) - V_{\pi,\gamma}(s_t))\right] \\
&= E_{\vec{s},\vec{a}\sim P_{\pi'}}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi,\gamma}(s_t,a_t)\right] \\
&= \sum_{t=0}^{\infty}\gamma^t E_{s\sim \Pr(s_t=s|\pi',M,s_0)} E_{a\sim\pi'(\cdot|s)}\left[A_{\pi,\gamma}(s,a)\right] \\
&= \frac{1}{1-\gamma} E_{s\sim d_{\pi',s_0}} E_{a\sim\pi'(\cdot|s)}\left[A_{\pi,\gamma}(s,a)\right]
\end{aligned}$$

where the last step uses the definition of d_{π',s_0}. □

The previous lemma leads to the following max norm bound, which is analogous to the Bellman error bound from Williams and Baird [1993] (theorem 5.1.3). This bound hides the important measure dependence on an optimal policy π^*.

COROLLARY 5.2.2. *Let M be an MDP.*

(Undiscounted) If T is finite, then for all policies π and for all s_0,

$$V_\pi(s_0) \geq V^*(s_0) - T||A_{\pi,t}||_\infty$$

(Discounted) If T is infinite, then for all stationary policies π, and for all s_0 and γ,

$$V_{\pi,\gamma}(s_0) \geq V^*_\gamma(s_0) - \frac{1}{1-\gamma}||A_{\pi,\gamma}||_\infty$$

5.3. Non-stationary Approximate Policy Iteration

The previous lemma avoided the use of the max norm error. Unfortunately, it is not clear how to use this result to provide more powerful bounds for the γ-discounted approximate algorithms of chapter 3, where we argued that the max norm error was appropriate. The underlying difficulty in applying this lemma is due to the fact that these *stationary* algorithms are not directly attempting to minimize the advantages.

This section presents the undiscounted *non-stationary* approximate policy iteration (NAPI) algorithm, where the errors made by the algorithm are directly related to the advantages, which allows us to avoid statements with max norms.

Recall from chapter 2 that for the undiscounted case, exact value iteration and exact policy iteration are identical (unlike in the discounted case), since the vectors computed in value iteration are equal to the value of the policy. However, there is a sensible policy iteration variant of the algorithm for the approximate setting.

5.3.1. The Algorithm. For simplicity, we only specify the algorithm for deterministic policies. First, let us define a **decision rule** to be a procedure for action selection at a *specified* time (see Puterman [1994]). We use this terminology to stress the distinction between a procedure for acting over the entire T-steps in the MDP (a *policy*) and a procedure for acting at a particular time (a *decision rule*). For a particular time t, the distribution $h(\cdot|s)$ over actions is a decision rule. A deterministic decision rule is of the form $h(s)$. A policy π for a T-epoch MDP M is a sequence of T decision rules $\{\pi(\cdot|s,0), \pi(\cdot|s,1), \ldots, \pi(\cdot|s,T-1)\}$.

The approximate algorithm assumes access to a PolicyChooser algorithm which takes as input a (T-epoch) deterministic policy π and a time t and returns a deterministic decision rule h. Informally, when given as input a policy π and a time t, the goal of the PolicyChooser is to construct the best decision rule for timestep t, subject to the constraint that π is followed for the other timesteps. An ideal PolicyChooser is one that returns the decision rule $h(s) = \arg\max_a Q_{\pi,t}(s,a)$.

The undiscounted NAPI algorithm is shown in algorithm 6. First, randomly initialize a T-epoch (deterministic) policy $\tilde{\pi} = (\tilde{\pi}(\cdot,0), \tilde{\pi}(\cdot,1), \ldots \tilde{\pi}(\cdot,T-1))$. Recursively updating backward from time $T-1$, the algorithm calls the PolicyChooser to obtain a decision rule $h_t(s)$, and then the algorithm simply sets $\tilde{\pi}(\cdot,t)$ to be h_t. The algorithm is a "policy iteration" algorithm (rather than a "value iteration" algorithm) since PolicyChooser is provided with the policy $\tilde{\pi}$ as an input (instead of a backed up approximation to the value of this policy).

Algorithm 6 T-Step NAPI

(1) Randomly initialize $\tilde{\pi} = (\tilde{\pi}(\cdot,0), \ldots \tilde{\pi}(\cdot,T-1))$
(2) For $t = T-1, \ldots 1$

$$\begin{aligned} h_t &= \text{PolicyChooser}(\tilde{\pi}, t) \\ \tilde{\pi}(\cdot,t) &= h_t \end{aligned}$$

(3) Return $\tilde{\pi}$

In the most general setting, the PolicyChooser has access to some sort of sampling model for obtaining information about the MDP (such as a generative model). If the state space is finite, then with access to a generative model, the PolicyChooser could provide an arbitrarily near-optimal policy at every step using sufficiently many calls at each state-action (as in the phased value iteration algorithm in section 2.5). However, this procedure clearly has large sample complexity for large state spaces and is not applicable to infinite state spaces. Ideally, we desire the PolicyChooser to generalize from a relatively small number of samples. These sample complexity issues are examined further in the next chapter.

5.3.2. The PolicyChooser Errors and the Advantages. An ideal PolicyChooser algorithm is one which returns the decision rule $\arg\max_a Q_{\pi,t}(s,a)$ upon input of a policy π and a time t. Let us now examine how the errors made by an arbitrary PolicyChooser algorithm effect the quality of the output policy of NAPI.

Let π be the input policy to PolicyChooser during update t and let h_t be the output of the policy chooser. The sensible definition for the **per state error** $\varepsilon_t(s)$ at update t is

$$\varepsilon_t(s) \equiv \max_{a \in \mathcal{A}} Q_{\pi,t}(s,a) - Q_{\pi,t}(s, h_t(s)) .$$

Intuitively, this error $\varepsilon_t(s)$ is the amount by which any action could improve upon the decision rule $h_t(s)$. If the decision rule $h_t(s)$ is greedy with respect to $Q_{\pi,t}(s,a)$, then this error is 0 at state s and clearly there is no better action when we are constrained to follow π for the remaining $t-1$ steps.

The following simple lemma shows that this error bounds the advantages of the *output* policy of NAPI. This lemma is useful in the following chapters. Importantly, note that this error ε_t was defined with respect to the *input* policy π to PolicyChooser at update t.

LEMMA 5.3.1. *Let $\tilde{\pi} = (h_0, \ldots, h_{T-1})$ be the output policy of NAPI . If π was the input policy to the PolicyChooser for the t-th update, then*

$$Q_{\tilde{\pi},t}(s,a) = Q_{\pi,t}(s,a)$$

Furthermore, if the per state error is $\varepsilon_t(s)$, then

$$A_{\tilde{\pi},t}(s,a) \leq \varepsilon_t(s)\,.$$

The key to the proof is that the state-action value $Q_{\pi,t}(s,a)$ does not depend on the choice of the initial action. Therefore, the choice of the decision rule $h_t(\cdot)$ does not alter $Q_{\pi,t}(s,a)$. The first result directly implies the error bound.

PROOF. Let π be the input to the PolicyChooser at update t. By definition of the t state-action value, $Q_{\pi,t}(s,a)$ does not depend on the initial action. Therefore, $Q_{\pi,t}(s,a)$ is not altered after update $t+1$, since after this update, the decision rules after time t are fixed to $h(\cdot, t+1), \ldots, h(\cdot, T-1)$. It follows that for the output policy $\tilde{\pi}$,

$$Q_{\tilde{\pi},t}(s,a) = Q_{\pi,t}(s,a)$$

and that

$$\begin{aligned} A_{\tilde{\pi},t}(s,a) &= Q_{\tilde{\pi},t}(s,a) - V_{\tilde{\pi},t}(s) \\ &= Q_{\tilde{\pi},t}(s,a) - Q_{\tilde{\pi},t}(s,h_t(s)) \\ &= Q_{\pi,t}(s,a) - Q_{\pi,t}(s,h_t(s)) \\ &\leq \varepsilon_t(s) \end{aligned}$$

where we have used the definition of ε_t. □

Importantly, this lemma shows how NAPI can control the size of the advantages since the errors of the PolicyChooser directly bound the advantages of the output policy. In the Remarks section of this chapter, it is pointed out that it is not clear how to obtain such a result for stationary (γ-discounted) algorithms. This is because there is no simple relation between the errors and the advantages of the output policy.

5.3.3. Regression and The Policy Chooser. The implications of this result are perhaps most clear in a function approximation setting. This section examines implementing the PolicyChooser algorithm with a regression algorithm and compares the results with that of chapter 3. There is a direct relation between the per state-action error defined above and the regression error.

In the next chapter, we consider a more natural policy search implementation of the PolicyChooser, which has no recourse to function approximation. In this policy search setting, we explicitly consider sample complexity bounds. In the current regression setting, we do not address the sample complexity for finding a good policy, though much work has gone

into understanding the sample complexity of regression algorithms which use some sort of average error (see Anthony and Bartlett [1999] for review).

The straightforward RegressionPolicyChooser algorithm is shown in algorithm 7. Similar to the function approximation methods in chapter 3, the algorithm approximates the state action values $Q_{\pi,t}$ with $\tilde{Q}_{\pi,t}$ for a particular input time t. The output decision rule is then greedy with respect to $\tilde{Q}_{\pi,t}$.

Algorithm 7 RegressionPolicyChooser(π, t)

(1) Approximate $Q_{\pi,t}$ with $\tilde{Q}_{\pi,t}$
(2) Set
$$\tilde{h}_t(s) = \arg\max_{a \in \mathcal{A}} \tilde{Q}_{\pi,t}(s,a)$$
(3) Return $\tilde{h}_t$

Define the **per state regression error** $\tilde{\varepsilon}_t(s)$ at the t-th update as

$$\tilde{\varepsilon}_t(s) \equiv \max_{a \in \mathcal{A}} \left| Q_{\pi,t}(s,a) - \tilde{Q}_{\pi,t}(s,a) \right| .$$

where π is the input policy to RegressionPolicyChooser at time t. The following theorem shows that the performance bound of NAPI using a RegressionPolicyChooser is dependent on an *average error* not a max norm error.

THEOREM 5.3.2. *Assume that NAPI uses RegressionPolicyChooser. Let π be the policy returned by NAPI with a regression error of $\tilde{\varepsilon}_t$. For all policies π' and for all states s_0,*

$$V_\pi(s_0) \geq V_{\pi'}(s_0) - 2T E_{s,t \sim d_{\pi',s_0}} \left[\tilde{\varepsilon}_t(s) \right] .$$

Hence, the amount π differs from some optimal policy π^* is dependent on the *average error* under a distribution that is provided by the future state distribution of the optimal policy π^*. Note that this is a significantly different situation than the bounds for γ-discounted approximate policy and value iteration (see section 3.2), where we argued the max norm bounds were appropriate. Here, we have the intuitive and appealing result that we desire our error to be small under the states visited by an optimal (or near-optimal) policy. Furthermore, note that there is only one factor of T in the bounds as opposed to the bounds for approximate discounted value and policy iteration where two factors of the horizon time are present.

The proof involves showing that the per-state update error $\varepsilon_t(s)$ is bounded by twice the regression error $\tilde{\varepsilon}_t(s)$.

PROOF. By definition of the RegressionPolicyChooser, the output h_t is greedy with respect to $\tilde{Q}_{\pi,t}$, and so for all actions a, $\tilde{Q}_{\pi,t}(s, h_t(s)) \geq \tilde{Q}_{\pi,t}(s,a)$. Let π be the input policy to RegressionPolicyChooser. By definition of $\varepsilon_t(s)$,

$$\begin{aligned}
\varepsilon_t(s) &= \max_{a \in \mathcal{A}} Q_{\pi,t}(s,a) - Q_{\pi,t}(s, h_t(s)) \\
&\leq \max_{a \in \mathcal{A}} \left(Q_{\pi,t}(s,a) - \tilde{Q}_{\pi,t}(s,a) + \tilde{Q}_{\pi,t}(s, h_t(s)) - Q_{\pi,t}(s, h_t(s)) \right) \\
&\leq 2 \max_{a \in \mathcal{A}} \left| Q_{\pi,t}(s,a) - \tilde{Q}_{\pi,t}(s,a) \right| \\
&= 2 \tilde{\varepsilon}_t(s) .
\end{aligned}$$

From the last lemma, it follows that for all a, $A_{\pi,t}(s,a) \leq \varepsilon_t(s) \leq 2\tilde{\varepsilon}_t(s)$. The performance difference lemma from the last section implies the result. □

Standard regression methods in the supervised learning literature typically use some distribution μ over the input space and attempt to minimize some average error with respect to μ. In our setting, the input space is $\{\text{states} \times \text{times}\}$. One scheme is to simply choose a distribution μ over state-times and try to minimize our error with respect to this μ. The previous theorem shows that the relevant "test" distribution is not μ but that of an optimal (or near-optimal) policy. Hence, it is sensible to try to choose a μ which matches the future state-time distribution of a good policy. It is not clear how to justify such a measure for the standard function approximation methods in chapter 3, where we argued the max norm was the appropriate error measure.

The important observation is that under this algorithm the reinforcement learning problem can be viewed as a supervised learning problem where we do not necessarily know the "test" distribution. We focus more on algorithms which use a distribution μ over state-times in the next chapter, where we explicitly examine the sample complexity of these approaches. It is difficult to get a handle on the sample complexity of this regression approach, due to the fact that accurately approximating the value function is only a means to the end goal of obtaining a good policy. The *classification* setting that we consider in the next chapter is a more natural supervised learning problem, since one could consider the decision rule $h_t(s)$ to be a "classifier" of actions.

5.4. Remarks

5.4.1. Stationarity and The Problem Of Greedy Updates. The avoidance of the max norm error in NAPI is inherently tied to the use of non-stationary policies. The reason is due to the direct control we have over the advantages as shown in lemma 5.3.1.

Let us step back and examine exact undiscounted value iteration. After the t-th step of exact value iteration, the maximal advantages at time t are set to 0, *ie* $\max_{s,a} A_{\pi,t}(s,a) = 0$. Furthermore, in the approximate version, lemma 5.3.1 shows that, regardless of the current policy π at update $t-1$, NAPI has the opportunity to set the maximal advantages with respect to time t to 0 by acting greedily with respect to $Q_{\pi,t}(s,a)$.

Such a result is not true even for *exact* γ-discounted, stationary policy iteration. Let π' be the result of an exact greedy update with π, *ie* $\pi'(s) = \arg\max_a Q_{\pi,\gamma}(s,a)$. After this exact update, there are no advantages that are necessarily set to 0. Chapter 7 considers the use of stationary policies in approximate updates, which is a considerably more involved analysis.

5.4.2. The Mismatched Measure for Gradient Methods. Let us consider the discounted case. In terms of the advantages, we can write the policy gradient (see section

4.2.2) as:

$$\begin{aligned}
\nabla V_\pi(s_0) &= \frac{1}{1-\gamma} E_{s\sim d_{\pi,s_0}} \left[\sum_a Q_\pi(s,a) \nabla \pi(a|s,\theta) \right] \\
&= \frac{1}{1-\gamma} E_{s\sim d_{\pi,s_0}} \left[\sum_a (A_\pi(s,a) + V_\pi(s)) \nabla \pi(a|s,\theta) \right] \\
&= \frac{1}{1-\gamma} E_{s\sim d_{\pi,s_0}} \left[\sum_a A_\pi(s,a) \nabla \pi(a|s,\theta) + V_\pi(s) \sum_a \nabla \pi(a|s,\theta) \right] \\
&= \frac{A}{1-\gamma} E_{s\sim d_{\pi,s_0}} E_{a\sim \text{Uniform}} \left[A_\pi(s,a) \nabla \pi(a|s,\theta) \right]
\end{aligned}$$

where we have used the fact for a probability distribution π, $\nabla \sum_a \pi(a|s,\theta) = \nabla 1 = 0$. Clearly, this expectation is with respect with to the future state distribution of the current policy π.

From the performance difference lemma 5.2.1, the difference between the performance of π and that of an optimal policy π^* is:

$$\frac{1}{1-\gamma} E_{s\sim d_{\pi^*,s_0}} E_{a\sim\pi^*} \left[A_\pi(s,a) \right] .$$

This elucidates the "mismeasure" problem for gradients. The gradient is small when the advantages are small under the *current* distribution d_{π,s_0}. However, to be close to optimal, the advantages must be small under d_{π^*,s_0}.

CHAPTER 6

μ-Learnability

This chapter considers the sample complexity of reliably choosing a good policy among some restricted class of policies Π in a large or infinite state MDP. The framework considered is one in which we have access to a sampling model that allows us to observe transitions in a T-epoch MDP M. The question that is studied is: how many transitions must be observed in order to have a sufficient amount of experience in order choose a "reasonably good" policy among Π?

The answer to this question clearly depends on what constitutes a "reasonably good" policy. The most straightforward goal is to find a policy that has return close to the highest return among those policies within Π. This is the goal of the trajectory tree method of Kearns, Mansour, and Ng [2000]. This algorithm uses a number of observed transitions that is *exponential* in T but has *no dependence* on the size of the state space (and has a natural dependence on the complexity of Π).

Clearly, the drawback in obtaining a practical algorithm is the exponential dependence on T. With practical concerns in mind, one could attempt to find a policy π that satisfies a more restricted notion of optimality in the hope that this allows us to obtain a polynomial sample complexity dependence on T. This motivates us to consider optimizing a policy with respect to a probability measure μ over the state space. The results from the last chapter showed that for a policy π to compete favorably against the performance of a policy π', then π only needs to have advantages that are small on average, with respect to the set of states that π' tends to visit. This suggests that imposing a measure μ over the state space is as a natural means to incorporate domain knowledge as to which states are important to optimize the performance at.

Informally, the goal here is to obtain a policy which competes favorably against those policies $\pi \in \Pi$ whose future state distribution is comparable to μ. The sample complexity question of interest is: how much experience is required to obtain a good policy with respect to our choice of μ?

This chapter presents the μ-PolicySearch algorithm which satisfies a more restricted notion of optimality based on the measure μ and which requires significantly less experience — μ-PolicySearch has a sample complexity bound that is now only *polynomial* in T and still has no dependence on the size of the state space. Importantly, although the algorithm gathers sufficiently less experience, the dependence on the complexity of the class Π is comparable to that of the trajectory tree method, which suggests that this method is making efficient use of samples.

From personal communication, Drew Bagnell and Andrew Ng are working on very similar algorithms.

These results suggest that using a measure μ might lead to feasible algorithms for problems which have both large-state spaces and large horizon times. Before heading in to the analysis, let us discuss the issue of efficient learning and what we hope to do by imposing the measure μ.

6.0.3. Efficient Use of Samples. The efficient reuse of samples in reinforcement learning is a considerably more tricky issue than in supervised learning. In supervised learning, we desire to learn some function $f(x)$ given a sample set of the form $\{(x, f(x))\}$. A common framework for supervised learning is one in which we have a "hypothesis class" $\mathcal{H}$ and desire to find a function $h \in \mathcal{H}$ that is a good approximation to f. In the supervised learning setting, every sample $(x, f(x))$ provides feedback for *all* $h \in \mathcal{H}$. This permits us to reliably choose a good $h \in \mathcal{H}$ with a number of samples that is far less than the size of this hypothesis set $|\mathcal{H}|$. For the case in which $|\mathcal{H}|$ is finite, we have $O(\log |\mathcal{H}|)$ bounds (ignoring other parameters) on the sample size sufficient to choose a near best $h \in \mathcal{H}$. For the case that $|\mathcal{H}|$ is infinite, sample complexity bounds are often stated in terms of some measure of the *complexity* of $\mathcal{H}$, such as the VC dimension (see Anthony and Bartlett [1999] for review). Crucially, these sample complexity bounds have *no dependence* on the size of the input domain.

Kearns, Mansour, and Ng [2000] provide the "trajectory tree" framework which generalizes these sample complexity results to reinforcement learning. The setting considered is where the goal is to find a policy in Π with performance near to the best policy in Π. A naive method to find a good policy is to simulate each policy in Π, which requires $O(\Pi)$ samples. This is clearly inefficient and is not applicable if Π is an infinite class.

The trajectory method assumes access to a generative model and builds a set of trees over all possible actions over the horizon time, requiring $O(A^T)$ calls. Efficient reuse of experience is possible since each tree simultaneously provides an estimate of the value of *all* $\pi \in \Pi$. This leads to the important $O(\log |\Pi|)$ sample size in order to choose a good $\pi \in \Pi$, with no dependence on the size of the state space (though the dependence on horizon time T is exponential). For the case of infinite Π, Kearns, Mansour, and Ng [2000] show how the standard VC complexity approaches can be applied.

6.0.4. A Measure μ and Exploration. This harsh exponential dependence on T can be viewed as the cost of exploration. By building a tree, we obtain sufficient information about the MDP to accurately estimate the values of all $\pi \in \Pi$ regardless of which states these policies tend to visit. Importantly, as with the related spare sampling algorithm (see section 2.5), this information is not enough to construct an accurate model of the MDP's transition probabilities.

We seek to avoid this A^T dependence. Clearly, this means that our planning algorithm will have significantly less information about the MDP. The question is then how should we collect and use this limited information about the MDP. The question of data collection is essentially a problem of exploration. As discussed in chapter 4, obtaining information about the MDP in an "on-policy" manner is in general not sensible, since ultimately it might be necessary to improve the policy at states where the current policy tends to visit infrequently.

We might hope that prior domain knowledge can help us deal with the problem of exploration. In this chapter, we consider optimizing the policy with respect to a probability measure μ over the state-space. The choice of μ is a means to incorporate prior knowledge

into the algorithm. The high level idea is to use this measure μ to obtain samples instead of building a tree or using an "on-policy" distribution. The tradeoff we seek is to obtain a polynomial T dependence in exchange for a more limited notion of optimality based on the measure μ.

In many challenging domains, it is clear that significant prior knowledge must be taken into account in order to obtain powerful algorithms. In supervised learning, this is often addressed in the choice of the hypothesis class $\mathcal{H}$ and the analogous incorporation of prior knowledge for supervised learning is in the choice of Π. A more unique aspect of reinforcement learning is knowledge related to where a good policy tends to visit. The trajectory tree method and gradient methods do not incorporate such knowledge. In many natural domains, we might have prior knowledge of the states a good policy tends to visit. In robotic control problems often we have an idea of the desired trajectory. In a queuing networks, there is rich literature on understanding the stability and the stationary properties of various controlled processes, and often there exists significant knowledge as to which operating regimes are appropriate. We desire our optimization algorithm to make use of such knowledge.

This chapter introduces the μ-PolicySearch algorithm, which is variant of NAPI. This variant uses a PolicyChooser algorithm which picks a decision rule from a "hypothesis set" Π_1. The policy returned is a non-stationary policy composed of a sequence of T decision rules chosen from Π_1. Note that each $h \in \Pi_1$ is just a mapping from states to actions, and so h can be viewed as a *classifier* of actions. The problem faced by the PolicyChooser is essentially one in which in it attempts to minimize a *cost sensitive classification loss function* where the training set distribution is obtained using the distribution μ.

The guarantee of μ-PolicySearch is that it drives the advantages to be small on *average* with respect to μ and this translates into a bound on the quality of the returned policy. The main results of μ-PolicySearch are:

- no dependence on the size of the state space
- polynomial sample complexity bounds on T
- efficient reuse of data (which allows infinite policy classes to be considered)
- a reasonable (restricted) notion of optimality based on μ

Throughout this chapter, we only deal with *non-stationary* policies in the T-epoch case. It turns out that obtaining similar guarantees using *stationary policies* (in the γ-discounted case) is much more challenging and this is addressed in the next chapter. For simplicity, this chapter only deals with deterministic policy classes Π and Π_1.

6.1. The Trajectory Tree Method

Let us begin by reviewing the trajectory tree method of Kearns, Mansour, and Ng [2000]. Assume that Π is some restricted, finite class of *deterministic* T-epoch policies, which is analogous to our "hypothesis" set. The question of interest is how many calls to a generative model are sufficient in order to find a policy that has return near to that of the best policy in Π, with respect to some start state s_0. A naive method to do this is to make $O(|\Pi|)$ calls in order to independently estimate the return of each $\pi \in \Pi$. However, this is clearly inefficient and is not applicable for infinite $|\Pi|$.

The trajectory tree method efficiently calls the generative model in order to find a good policy in Π. The idea is to start at state s_0 and build a tree recursively, by trying each

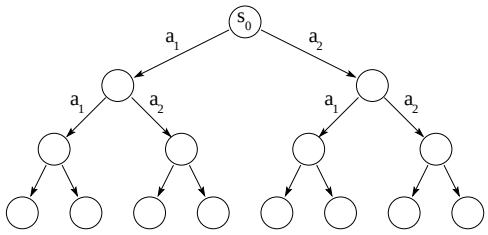

FIGURE 6.1.1. A Trajectory Tree. See text for description.

action once at each state encountered as shown in figure 6.1.1. At the root s_0, all actions are sampled *once* using the generative model. Then for each child, we also sample each action once. This is continued until the tree is completely filled out to depth T, requiring $O(A^T)$ samples for each tree.

It is straightforward to construct an unbiased estimate of the return of a deterministic policy π using a single tree τ, since π defines a unique trajectory on the tree. Let $R(\pi, \tau)$ be the average return on the trajectory defined by π on the tree τ. $R(\pi, \tau)$ provides an unbiased estimate of $V_\pi(s_0)$. Now consider building m trees $\tau_1, \ldots, \tau_m$. The obvious empirical estimate of π is

$$\hat{V}_\pi(s_0) = \frac{1}{m} \sum_{i=1}^{m} R(\pi, \tau_i) \, .$$

which can be used to simultaneously estimate the value of each $\pi \in \Pi$. The question that is addressed in the following subsections is: what is the appropriate value of m in order to have a *uniform convergence* result, *ie* in order for $\hat{V}_\pi(s_0)$ to accurately approximate $V_\pi(s_0)$ for all $\pi \in \Pi$.

Before doing this, let us digress by recalling the sample complexity of the sparse sampling algorithm of Kearns, Mansour, and Ng [1999] (see section 2.5 for a review of this method). Here, the algorithm itself is the policy, since the algorithm takes as input a state and then outputs an action. For *each* input, the algorithm builds a tree using the generative model in order to compute a *single* action (though the tree here is slightly different as shown in figure 2.5.1).

The sparse sampling algorithm/policy is an ε near-optimal policy and makes $\left(\frac{AT}{\varepsilon}\right)^{O(T \log \frac{T}{\varepsilon})}$ calls to the generative model *per input*. This of course leads to the question of why bother building trajectory trees to search a restricted class of policies when we could just use the generative model to execute a near-optimal policy, since both have sample complexity bounds that have exponential dependence on T (though this dependency is per timestep for the sparse sampling algorithm).[1]

In the original formulation, this method was applied to partially observable MDPs. Since sparse sampling methods cannot be directly applied to the partially observable setting, this search is sensible. Also, in the MDP setting, if we don't have access to the generative model at runtime, then it might be desirable to find a good policy offline by searching Π.

[1] It should be noted that the sample complexity bound for the sparse sampling algorithm is still worse though they are both exponential algorithms. Essentially, we are comparing "$O((mA)^T)$" vs. "$O(mA^T)$", where "m" is polynomial in T for both algorithms.

However, the point here is not to justify the use of this method (which is clearly provided by Kearns, Mansour, and Ng [2000]), but to recognize (for the MDP setting) that although this exponential dependence allows us to efficiently *search* a restricted class of policies, we can execute an (*unrestricted)* ε near-optimal policy with a dependence that is also exponential in T.

6.1.1. The Case of Finite Π. As discussed earlier, a hallmark of supervised learning methods is a $\log$ dependence on the size of the hypothesis space. The following straightforward theorem shows that this result can be replicated in the reinforcement learning setting.

THEOREM 6.1.1. *(Finite Π; Kearns, Mansour, and Ng [2000]) Let Π be a deterministic class of policies for a T-epoch MDP M and let $\hat{V}_\pi(s_0)$ be the empirical value function constructed from m trees. With an appropriate choice of m, the total number of calls made to the generative model is*

$$O\left(\frac{A^T}{\varepsilon^2}\log\frac{|\Pi|}{\delta}\right)$$

and with probability greater than $1-\delta$, the estimates $\hat{V}_\pi(s_0)$ satisfy the following accuracy condition for all $\pi \in \Pi$,

$$|V_\pi(s_0) - \hat{V}_\pi(s_0)| \leq \varepsilon\,.$$

PROOF. For any $\pi \in \Pi$, each tree provides an unbiased, independent estimate of the $V_\pi(s_0)$. By Hoeffding's bound and the union bound, we have that the probability there exists a π such that $|\hat{V}_\pi(s_0) - V_\pi(s_0)| > \varepsilon$ is less than $|\Pi|\exp(-2m\varepsilon^2)$. This implies that $m = O(\frac{1}{\varepsilon^2}\log\frac{|\Pi|}{\delta})$ is sufficient to obtain a δ bound on this probability. The result follows since each tree requires $O(A^T)$ calls to the generative model. □

6.1.2. The Case of Infinite Π. For simplicity, we only review the $A = 2$ binary action case (and extensions to the multi-action case can be found in Kearns, Mansour, and Ng [2002]). The analysis parallels the theme of efficient data re-use in supervised learning.

In the supervised learning setting, a crucial observation is that eventhough a hypothesis set $\mathcal{H}$ maybe infinite, the number of possible behaviors of $\mathcal{H}$ on a finite set of states is not necessarily exhaustive. Let us review the usual definition of the VC dimension for a hypothesis set $\mathcal{H}$ of boolean functions. We say that the set $x_1, x_2, \ldots x_d$ is shattered if there exists an $h \in \mathcal{H}$ that can realize any of the possible 2^d labellings. The **VC dimension VC**$(\mathcal{H})$ is the size of the largest shattered set. It is known that if $d = \mathrm{VC}(\mathcal{H})$, then the number of possible labellings $\Phi_d(m)$ on a set of m points by functions in $\mathcal{H}$ is at most $(\frac{em}{d})^d$. For $d \ll m$, this is much less than 2^m. This bound provides the backbone for proving classical uniform convergence results.

These ideas also lead to uniform convergence results for the infinite Π case. For any particular tree, a policy π induces a set of "labellings" on the tree, where the label at each node/state is just the action chosen by π at that node. By definition, $R(\pi, \tau)$ is solely determined by the labelling of the tree τ by π and the reward function. Therefore if two policies have different values of $\hat{V}$, then there must exist a node in one of the m trees that is labelled differently by the two policies. Hence, the number of different values of $\hat{V}_\pi$ for the policies $\pi \in \Pi$ is bounded by the number of different labellings on the m trees by Π.

Again, the key observation is that although the set Π may be infinite, the set of possible labellings for the m trees is not exhaustive. Note that each $\pi \in \Pi$ is a deterministic

mapping from the state space to a set of two actions, and so π can be viewed as a boolean function. Hence, let $\mathrm{VC}(\Pi)$ denote the VC dimension of the set of Boolean functions in Π. For m trees, there are $m2^T$ nodes and so the number of possible labellings on these trees is $\Phi_d(m2^T) \leq (\frac{em2^T}{d})^d$. The formalized argument leads to the following theorem.

THEOREM 6.1.2. *(Infinite Π; Kearns, Mansour, and Ng [2000]) Let Π be a deterministic class of policies for a binary action T-epoch MDP and let $\hat{V}_\pi(s_0)$ be the empirical value function constructed from m trees. With an appropriate choice of m, the total number of calls made to the generative model is*

$$O\left(\frac{2^T}{\varepsilon^2}(T\mathrm{VC}(\Pi) + \log\frac{1}{\delta})\right)$$

and with probability greater than $1-\delta$, the estimates $\hat{V}_\pi(s_0)$ satisfy the following accuracy condition for all $\pi \in \Pi$,

$$|V_\pi(s_0) - \hat{V}_\pi(s_0)| \leq \varepsilon\,.$$

The proof is somewhat technical and is not provided here.

6.1.3. Approximate Planning and PEGASUS. These uniform convergence results immediately imply the following corollary.

COROLLARY 6.1.3. *For the case considered in theorem 6.1.1 or 6.1.2. Let*

$$\pi = \arg\max_{\pi' \in \Pi} \hat{V}_{\pi'}(s_0)\,.$$

Then with probability greater than $1-\delta$, for all $\pi' \in \Pi$,

$$V_\pi(s_0) \geq V_{\pi'}(s_0) - 2\varepsilon\,.$$

Hence, the optimization problem is just a search problem over $\hat{V}_\pi(s_0)$. Unfortunately, this might be a rather formidable task since the size of the trees are exponential in T.

Before we examine the complexity of this search, let us now review the PEGASUS method of Ng and Jordan [2001]. This method can viewed as providing a compact representation of the trajectory tree. The algorithm assumes access to a **deterministic generative model**, which is stronger assumption than access to a generative model. Roughly speaking, it assumes that we have an implementation of a generative model that has no internal random number generator and that in addition to providing the generative model with a state-action as input, we must also provide it with a random number (in order for it draw samples from the transition probability). In many problems, such as those where we implement the generative model on our computer, this is quite a reasonable assumption, since we often have to explicitly use a random number generator to induce stochasticity.

The key insight to this method is that if we fix the seed to our random number generator, then this uniquely determines a trajectory tree. Hence, to represent one trajectory tree, we do necessarily have the memory problem of generating an entire tree and storing it. The tree can be compactly represented by the seed to the random number generator. Though we have a concise representation of the tree, we still have the computational cost of computing a transition (using the seeded random numbers). Here, *the sample complexity question now becomes one of computational complexity* (as discussed in introduction to this thesis, see section 1.2).

The most common optimization method is to perform a local search to maximize $\hat{V}_\pi(s_0)$. Unfortunately, it turns out that the exploration problem still rears its head in this computational problem. Let us return to example 4.3.1 from chapter 4, where the agent desires to reach a goal state in a 50 state MDP (two actions move the agent to the left and one action moves the agent to the right). Building a tree is clearly not feasible since the required sample size is roughly 10^{25}, though PEGASUS allows us to avoid building this tree. However, once again we have the same problem discussed in chapter 4, which is that with a randomly initialized policy, then the number of calls to the generative model needed to obtain a non-zero gradient is exponential in T. Hence, the number of transitions that we must compute is still exponential in T. Essentially, PEGASUS is a variance reduction mechanism not a means to solve exploration.

In general, the lower bound for the sparse sampling algorithm (in section 2.5) suggests that the factor of A^T is unavoidable (since there could be some leaf of the tree that we must discover in order to find our good policy, see subsection 2.5.3). Therefore, a tradeoff is to be expected if we desire an algorithm with sample complexity that is polynomial in T.

6.2. Using a Measure μ

This factor of A^T does not take into account domain knowledge of where good policies tend to visit. The performance difference lemma (5.2.1) of the last chapter quantified the importance of optimizing our policy at states where a good policy tends to visit. As an attempt to deal with this exploration problem through the use of prior knowledge, we introduce a particular distribution μ and optimize with respect to this distribution.

The μ-PolicySearch algorithm presents an interesting tradeoff — the factor of $O(A^T)$ can be reduced to a polynomial T bound under a restricted notion of optimality. In practice, we desire to tackle problems with both infinite state spaces and large horizon times. These results suggests that planning in such problems may be feasible by carefully considering how to choose μ through domain knowledge.

6.2.1. The μ-Reset Model. The generative model allows us to obtain samples from any state of our choosing. Let us now consider using a μ-reset model, which is an intermediate sampling model between the generative model and the online simulation model (see section 2.4).

Let $\mu(s, t)$ be a joint distribution over states and times. As with the future state distribution, assume that μ is uniform over the times in the set $\{0, \ldots, T-1\}$. A **μ-reset model** is defined as follows. The model allows simulation of the MDP in the usual way (as in the online simulation model of 2.4) and the model allows *resets*. If time t is given as an input to the model, the next state is set to $s' \sim \mu(\cdot|t)$. Essentially, the μ-reset model allows us to simulate the MDP and reset the state in the MDP according any of the T distributions, $\mu(\cdot|0), \mu(\cdot|1), \ldots \mu(\cdot|T-1)$.

This is a weaker assumption than having access to a generative model and a considerably weaker assumption than having complete knowledge of the transition matrix in M. In a large or continuous state space, it might be difficult to obtain two samples from exactly the same state (as is assumed by a generative model). For example, for a simulator that is instantiated by a physical model in the real world, it may be infeasible to reconfigure the system into the exact same state twice. Our assumption is that we only need our physical simulator to configure the states according to the same *distribution*. Also, note that we can

always simulate a μ-reset with a (deterministic or non-deterministic) generative model, but we can't necessarily simulate a generative model with a μ-reset model.

The μ-PolicySearch algorithm ties the optimization to the distribution μ and so optimality guarantees are dependent on this choice. It turns out that a sensible choice for μ is the future state distribution of a good policy. Additionally, if we desire to set all $\mu(\cdot|t)$ to a single distribution $\rho(\cdot)$ then it is sensible to choose $\rho(\cdot)$ to be the stationary distribution of a good policy.

Although a μ-reset model may be a weaker simulation assumption, we desire control over the choice of μ in order to select a "good" reset model. With access to a generative model, we have the option to simulate the μ-reset model of our choice.

6.2.2. Generalized Value Functions. The following overloaded definitions are useful. Recall from subsection 5.3.1 that a decision rule specifies an action selection procedure for just one timestep, while a policy specifies the means of acting over the entire T-epoch MDP.

DEFINITION 6.2.1. Let M be a T-epoch MDP, t be a time, π be a policy for M, and μ be a state-time distribution.

The **value** $V_{\pi,t}(\mu)$ of $\mu(\cdot)$ is:

$$V_{\pi,t}(\mu) \equiv E_{s\sim\mu(\cdot|t)}\left[V_{\pi,t}(s)\right] .$$

The **state-action value** $Q_{\pi,t}(\mu,h)$ of a decision rule h and $\mu(\cdot)$ is:

$$Q_{\pi,t}(\mu,h) \equiv E_{s\sim\mu(\cdot|t)}E_{a\sim h(\cdot|s)}\left[Q_{\pi,t}(s,a)\right] .$$

The **advantage** $A_{\pi,t}(\mu,h)$ of a decision rule h and $\mu(\cdot)$ is:

$$A_{\pi,t}(\mu,h) \equiv E_{s\sim\mu(\cdot|t)}E_{a\sim h(\cdot|s)}\left[A_{\pi,t}(s,a)\right] .$$

Note that $Q_{\pi,t}(\mu,h)$ represents the expected value of choosing an action $a\sim h(\cdot|s)$ when given a state $s\sim\mu(\cdot|t)$ and then following π for the remaining $t-1$ steps. Hence, the notation is overloaded with $Q_{\pi,t}(s,a)$. The value $V_{\pi,t}(\mu)$ and the advantage $A_{\pi,t}(\mu,h)$ have similar interpretations. Note the familiar equality:

$$A_{\pi,t}(\mu,h) = Q_{\pi,t}(\mu,h) - V_{\pi,t}(\mu) .$$

Under μ, a natural goal is to find a policy π such that there does not exist an h such that $A_{\pi,t}(\mu,h)$ is large. Intuitively, this goal is to find a policy π that has small advantages with respect to μ.

6.3. μ-PolicySearch

Recall from the performance difference lemma (5.2.1), that the difference in value between an optimal policy π^* and π at s_0 is

$$V_{\pi^*}(s_0) - V_{\pi}(s_0) \quad = \quad T E_{(s,t)\sim d_{\pi^*,s_0}} E_{a\sim\pi^*(\cdot|s,t)}\left[A_{\pi,t}(s,a)\right] .$$

Therefore, if *each* advantage of π is less than ε/T then π has value that is ε near-optimal.

In large or infinite state spaces, the sample complexity required to guarantee that each advantage is small could be excessive. Instead, let us consider forcing the average of the advantages to be small with respect to μ.

The μ-PolicySearch algorithm is a "policy search" variant of NAPI (see subsection 5.3.1) which uses a restricted class of deterministic decision rules Π_1. This Π_1 is analogous to our "hypothesis" class $\mathcal{H}$. At time t of the algorithm, the μ-PolicyChooser attempts to find a good decision rule $h \in \Pi_1$ and this h is then used to set $\pi(\cdot, t)$.

The class Π_1 induces the class of T-epoch policies

$$\Pi \equiv \Pi_1 \times \Pi_1 \times \ldots \times \Pi_1 = \Pi_1^T$$

where each $\pi \in \Pi$ is a sequence of T decision rules, *ie* for each t, $\pi(\cdot, t) \in \Pi_1$. Note that the policy returned by μ-PolicySearch is in Π.

Our goal is to find a policy π such that for all times $t < T$, and for all $h \in \Pi_1$.

$$A_{\pi,t}(\mu, h) \leq \frac{\varepsilon}{T}.$$

Intuitively, this condition states that at each time t, there is no $h \in \Pi_1$ that has a large advantage over π on *average* with respect to $\mu(\cdot|t)$. Interestingly, note that this condition does not necessarily imply that the future distribution of π is similar to μ.

6.3.1. μ-Optimality. The following lemma gives us reassurance as to why this is a sensible goal. If p and q are two distributions over a finite (or countable) set $\mathcal{X}$, we use the standard definition that $||p - q||_1 = \sum_{x \in \mathcal{X}} |p(x) - q(x)|$, and if $\mathcal{X}$ is a continuous space, the sum is replaced by an integral. Recall, $\Pi = \Pi_1^T$.

THEOREM 6.3.1. *(μ-Optimality) Let M be a T-epoch MDP and let Π_1 be a set of decision rules for M. Assume that a policy π satisfies for all $h \in \Pi_1$ and $t < T$,*

$$A_{\pi,t}(\mu, h) \leq \frac{\varepsilon}{T}.$$

Then for all policies $\pi' \in \Pi$ and for all s_0,

$$V_\pi(s_0) \geq V_{\pi'}(s_0) - \varepsilon - T||d_{\pi',s_0} - \mu||_1.$$

Note that this guarantee holds for all s_0 as opposed to the trajectory tree method, which holds only for the prechosen s_0 (which was used for the root in the trees). Also note that the optimality guarantee of the trajectory tree method did not contain the penalty term $||d_{\pi',s_0} - \mu||_1$.

Essentially, the bound states that the policy π is guaranteed to compete favorably against any policy $\pi' \in \Pi$ whose future state distribution is close to μ. The bound is stated in terms of the additive mismatch between d_{π',s_0} and μ. It is straightforward to construct an alternative bound in terms of a multiplicative mismatch $||\frac{d_{\pi',s_0}}{\mu}||_\infty$, as was done in Kakade and Langford [2002].

This bound looks somewhat weak, since it suggests that we must choose μ rather carefully to match a good policy. However, it should be noted that this bound is essentially identical to the performance bound in the supervised learning setting where we "train" under one particular input distribution $\mu(x)$ and we are "tested" under a different distribution $P(x)$. In practice, this bound can be very loose, so we hope to achieve much better performance.

The more *general* statement is just that we know the advantages $\mathcal{A}_{\pi,t}(\mu, h)$ are $\frac{\varepsilon}{T}$ small with respect to Π_1 and that our performance regret is given by the performance difference lemma (5.2.1). The previous theorem provides the simplest performance bound using these

constraints. Unfortunately, there are few results on more informative and tighter bounds for this common setting.

The following proof implicitly uses the fact that our "training" and "testing" distributions are different. Here our "training" distribution is μ and our "test" distribution is d_{π',s_0}.

PROOF. Let $h_t(s) = \pi'(s,t)$ and so $h_t \in \Pi_1$. Using the performance difference lemma (5.2.1), we have

$$\begin{aligned} V_{\pi'}(s_0) - V_\pi(s_0) &= T E_{(s,t)\sim d_{\pi',s_0}} E_{a\sim\pi'(\cdot|s,t)}\left[A_{\pi,t}(s,a)\right] \\ &\leq T E_{(s,t)\sim\mu} E_{a\sim\pi'(\cdot|s,t)}\left[A_{\pi,t}(s,a)\right] + T||d_{\pi',s_0} - \mu||_1 \\ &= \sum_t E_{s\sim\mu(\cdot|t)} E_{a\sim\pi'(\cdot|s,t)}\left[A_{\pi,t}(s,a)\right] + T||d_{\pi',s_0} - \mu||_1 \\ &= \sum_t A_{\pi,t}(\mu,h_t) + T||d_{\pi',s_0} - \mu||_1 \\ &= \varepsilon + T||d_{\pi',s_0} - \mu||_1 \end{aligned}$$

where the last step follows by assumption. □

6.3.2. Main Results of μ-PolicySearch. First, let us specify an exact version of μ-PolicySearch which provides insight into the sample based version. Given a $t-1$ step policy π, the natural goal is to find an $h \in \Pi_1$ that optimizes $Q_{\pi,t}(\mu,h)$. Intuitively, $\mu(\cdot|t)$ is the distribution over states at time t that we wish to stress, so we desire to find a good decision rule with respect to this distribution. The Exact μ-PolicySearch algorithm (8) is a version of NAPI that does this optimization exactly.

Algorithm 8 Exact μ-PolicySearch

(1) Randomly initialize $\tilde{\pi}$
(2) For $t = T-1,\ldots,0$

$$\begin{aligned} h_t &= \arg\max_{h\in\Pi_1} Q_{\tilde{\pi},t}(\mu,h) \\ \tilde{\pi}(\cdot,t) &= h_t \end{aligned}$$

(3) Return $\tilde{\pi}$

The following theorem shows that this exact algorithm exactly achieves our goal with $\varepsilon = 0$.

THEOREM 6.3.2. *(Exact μ-PolicySearch) Let M be a T-epoch MDP and let Π_1 be a set of decision rules for M. Exact μ-PolicySearch returns a policy $\tilde{\pi}$ such that for all $h \in \Pi_1$ and $t < T$,*

$$A_{\tilde{\pi},t}(\mu,h) \leq 0\,.$$

PROOF. Let π be the input policy at update t to the ExactPolicyChooser in the algorithm. By construction, of h_t, it follows that $Q_{\pi,t}(\mu,h_t) \geq Q_{\pi,t}(\mu,h)$ for $h \in \Pi_1$. Lemma 5.3.1 shows that the state-action values of the output policy $\tilde{\pi}$ are identical to this

input policy π. Hence, for all $h \in \Pi_1$, $Q_{\tilde{\pi},t}(\mu, h_t) \geq Q_{\tilde{\pi},t}(\mu, h)$. Using the definition of $A_{\tilde{\pi},t}(\mu, h)$,

$$\begin{aligned} A_{\tilde{\pi},t}(\mu, h) &= Q_{\tilde{\pi},t}(\mu, h) - V_{\tilde{\pi},t}(\mu) \\ &= Q_{\tilde{\pi},t}(\mu, h) - Q_{\tilde{\pi},t}(\mu, h_t) \\ &\leq 0 \end{aligned}$$

Where we have used the fact that $\tilde{\pi}(\cdot, t) = h_t$. □

From the μ-optimality theorem (6.3.1), this π has value at s_0 that is $T||d_{\pi,s_0} - \mu||_1$ close to the best value in Π at s_0.

Algorithm 9 μ-PolicySearch(Π_1)

(1) Randomly initialize $\tilde{\pi}$
(2) For $t = T-1, \ldots, 0$

$$\begin{aligned} h_t &= \mu\text{ PolicyChooser}(\tilde{\pi}, t, \Pi_1) \\ \tilde{\pi}(\cdot, t) &= h_t \end{aligned}$$

(3) Return $\tilde{\pi}$

Of course, this exact algorithm is impractical, and we are interested in understanding the sample complexity of implementing this algorithm with a sample based PolicyChooser. Algorithm 9 presents a high level sketch of the μ-PolicySearch algorithm. The algorithm uses μ-PolicyChooser to construct the decision rules h_t. The μ-PolicyChooser uses only a μ-reset model to find a good decision rule among Π_1. First, let us provide the main theorem on sample complexity bounds for μ-PolicySearch to return a "reasonably good" policy.

THEOREM 6.3.3. *(μ-PolicySearch) Let M be a T-epoch MDP and let Π_1 be a class of deterministic decision rules for M. For any ε and δ, the total number of observed transitions by μ-PolicySearch is*

- *if Π_1 is finite,*

$$O\left(\frac{A^2T^4}{\varepsilon^2}(\log|\Pi_1| + \log\frac{T}{\delta})\right).$$

- *if Π_1 is infinite and M is a binary action MDP,*

$$O\left(\frac{T^4}{\varepsilon^2}(\mathrm{VC}(\Pi_1) + \log\frac{T}{\delta})\right).$$

For either case, with probability greater than $1-\delta$, μ-PolicySearch returns a policy π such that for all $h \in \Pi_1$ and $t < T$,

$$A_{\pi,t}(\mu, h) \leq \frac{\varepsilon}{T}.$$

Hence, with μ-PolicySearch, we can obtain our more restricted optimality guarantee with only *polynomial* T dependence while maintaining *no dependence* on the size of the state space. Importantly, the dependence on the complexity of Π_1 is comparable to that of the trajectory tree method, which suggests that this method also makes efficient use of samples.

Extensions to multi-action MDPs and stochastic policy classes are presented in Kearns, Mansour, and Ng [2000] and these results can be generalized to our setting as well.

The following sections spell out the algorithm and the technical lemmas required for the proof. We start by specifying the μ-PolicyChooser that is used. This algorithm is essentially a *cost sensitive classification* algorithm.

6.3.3. The μ-PolicyChooser. The μ-PolicyChooser attempts to return a good decision rule from Π_1 using only the μ-reset model. A naive and inefficient procedure is to independently estimate $Q_{\pi,t}(\mu, h)$ for each $h \in \Pi_1$ and use the empirical maximum. We turn to importance sampling for efficient estimation.

We can write this value as:

$$Q_{\pi,t}(\mu, h) = AE_{s\sim\mu(\cdot|t)}E_{a\sim\text{Uniform}}\left[Q_{\pi,t}(s,a)I(h(s)=a)\right]$$

where Uniform is the is the uniform distribution on the action space (of size A) and $I(h(s) = a)$ is the indicator function which is 1 if $h(s) = a$ and 0 else. In the Remarks section (6.4.1), we point out that this function is essentially a cost sensitive classification loss function for a "classifier" h with weights $Q_{\pi,t}$.

Algorithm 10 μ-PolicyChooser(π, t, Π_1)

(1) obtain m samples of the form $\{(s_i, a_i, \hat{Q}_i)\}$ where
   (a) $s_i, a_i \sim \mu(\cdot|t) \times$ Uniform
   (b) $\hat{Q}_i$ is an estimate of $Q_{\pi,t}(s_i, a_i)$

(2) define the function $\hat{Q}$

$$\hat{Q}_{\pi,t}(\mu, h) = \frac{A}{m}\sum_i \hat{Q}_i I\left(h(s_i) = a_i\right)$$

(3) determine $\hat{h}$

$$\hat{h} = \arg\max_{h\in\Pi_1} \hat{Q}_{\pi,t}(\mu, h)$$

(4) Return $\hat{h}$

The sampling procedure for the μ-PolicyChooser is as follows (see algorithm 10). At the t-th step, obtain an $s \sim \mu(\cdot|t)$ and an $a \sim$ Uniform. Then follow π for the remaining t steps to obtain an unbiased estimate of $\hat{Q}$ of $Q_{\pi,t}(s, a)$. An unbiased estimate $Q_{\pi,t}(\mu, h)$ is then $A\hat{Q}I(h(s) = a)$.

Consider obtaining m-samples constructed in this fashion. For the i-th sample, let (s_i, a_i) be the sample of the state-action and let $\hat{Q}_i$ be the estimate of $Q_{\pi,t}(s_i, a_i)$. Our unbiased estimate of $Q_{\pi,t}(\mu, h)$ is then

$$\hat{Q}_{\pi,t}(\mu, h) = \frac{A}{m}\sum_i \hat{Q}_i I\left(h(s_i) = a_i\right)$$

for any policy $h \in \Pi_1$. The total number of transitions observed by μ-PolicyChooser under this method is mt , since each estimate $\hat{Q}_i$ uses t transitions.

6.3.4. Uniform Convergence Results for μ-PolicyChooser. The following lemmas determine a value of m sufficient to obtain uniform convergence results for μ-PolicySearch. Recall that μ-PolicySearch makes T-calls to the μ-PolicyChooser and so there are T functions $\hat{Q}_{\pi,T-1}(\mu,\cdot),\ldots,\hat{Q}_{\pi,0}(\mu,\cdot)$ constructed. We desire each of these functions to be accurate to the tune of $\frac{\varepsilon}{T}$. Since each call to μ-PolicyChooser requires $O(mT)$ transitions, the total number of transitions observed by μ-PolicySearch is mT^2.

The result for the finite Π case is presented first.

LEMMA 6.3.4. *(Finite Π_1) Let Π_1 be a class of deterministic decision rules for a T-epoch MDP M. Let*

$$m = O\left(\frac{A^2T^2}{\varepsilon^2}(\log|\Pi_1| + \log\frac{1}{\delta})\right) .$$

Upon input of a policy π and time t, the μ-PolicySearch algorithm constructs a function $\hat{Q}_{\pi,t}(\mu,h)$ such that with probability greater than $1-\delta$, for all $h \in \Pi_1$

$$|\hat{Q}_{\pi,t}(\mu,h) - Q_{\pi,t}(\mu,h)| \leq \frac{\varepsilon}{T} .$$

Thus, we have the important $O(\log|\Pi_1|)$ dependence.

PROOF. The values $A\hat{Q}_i I(h(s_i) = a_i)$ are in the bounded interval $[0,A]$. Hoeffding's and the union bound imply that the probability that there exists an $h \in \Pi_1$ where $|\hat{Q}_{\pi,t}(\mu,h) - Q_{\pi,t}(\mu,h)| \geq \frac{\varepsilon}{T}$ is less than $|\Pi_1|\exp\left(-\frac{2m\varepsilon^2}{A^2T^2}\right)$. The result follows by setting this bound to be less than δ. □

The following theorem shows that μ-PolicySearch can be extended to infinite policy classes just as was done in the trajectory tree method.

LEMMA 6.3.5. *(Infinite Π_1) Let Π_1 be an infinite class of deterministic decision rules for a binary action T-epoch MDP M. Let*

$$m = O\left(\frac{T^2}{\varepsilon^2}(\mathrm{VC}(\Pi_1) + \log\frac{1}{\delta})\right) .$$

Upon input of a policy π and time t, the μ-PolicySearch algorithm constructs a function $\hat{Q}_{\pi,t}(\mu,h)$ such that with probability greater than $1-\delta$, for all $h \in \Pi_1$

$$|\hat{Q}_{\pi,t}(\mu,h) - Q_{\pi,t}(\mu,h)| \leq \frac{\varepsilon}{T} .$$

The proof is essentially identical to that in Kearns, Mansour, and Ng [2000], except we use importance sampling and we are in simpler case where the tree depth is one. For completeness the proof is provided.

For the proof, it is useful to define $VC_r(\mathcal{H})$ for a set $\mathcal{H} = \{h | h : X \to [-B,B]\}$ of real valued functions bounded by B. Define $VC_r(\mathcal{H})$ to be the standard VC-dimension of the set of all binary functions $\{I(h,r,\cdot) : h \in \mathcal{H}, r \in (-B,B)\}$, where $I(h,r,x) = 1$ if $h(x) \geq r$, $I(h,r,x) = 0$ else (as in Vapnik [1982]). Intuitively, this is the set of all indicators of all threshold functions constructed from $\mathcal{H}$.

PROOF. Note that each $h \in \Pi_1$ can be viewed as a set of real valued functions which map a sample $(s, a, \hat{Q})$ to a real value in $[0, 2]$, where the corresponding function value is $2I\left[h(s) = a\right]\hat{Q}$. Let us denote Π_1 by $\tilde{\Pi}_1$ when it is viewed as this set of real valued functions that map samples of the form $(s, a, \hat{Q})$ to $[-2, 2]$. First, let us prove that

$$VC_r(\tilde{\Pi}_1) = O(VC(\Pi_1))\,.$$

(see Kearns, Mansour, and Ng [2000] for a more general result for trajectory trees).

Let $d = VC(\Pi_1)$. The set Π_1 can realize $(\frac{em}{d})^d$ labellings for a set of m samples. Further, for any sample $(s, a, \hat{Q})$, there are only 2 possible values that this sample can take for all $h \in \tilde{\Pi}_1$ (either $2\hat{Q}$ or 0). Hence, there are only $2m$ values that functions in $\tilde{\Pi}_1$ could take on m samples, and so we only need to consider $2m$ settings for threshold parameter r.

Therefore, the set of indicator functions $\{I(h, r, \cdot)\}$ can realize at most at most $2m(\frac{em}{d})^d$ labellings on m samples. If this set is to shatter m samples, then it must be greater than 2^m, *ie*

$$2m(\frac{em}{d})^d \geq 2^m\,.$$

This implies that $m = O(d) = O(VC(\Pi_1))$. Since $m \geq VC_r(\tilde{\Pi}_1)$, $VC_r(\tilde{\Pi}_1) = O(VC(\Pi_1))$.

The other useful result from Vapnik [1982] is on the estimation accuracy. In our setting, this implies, with probability greater than $1 - \delta$,

$$\sup_{h \in \Pi_1} |\hat{Q}_{\pi,t}(\mu, h) - Q_{\pi,t}(\mu, h)| \leq O\left(\sqrt{\frac{d \log \frac{m}{d} + \log \frac{1}{\delta}}{m}}\right)$$

where $d = VC_r(\tilde{\Pi}_1) = O(VC(\Pi_1))$. The result now follows by substituting $m = O\left(\frac{T^2}{\varepsilon^2} \log \frac{VC(\Pi_1)}{\delta}\right)$ into the previous bound. □

Now we are ready to prove our theorem on sample complexity bounds of μ-PolicySearch

PROOF. (of theorem 6.3.3) The μ-PolicyChooser is called T times, so if $\delta \leftarrow \frac{\delta}{T}$ then all T functions $\hat{Q}_{\tilde{\pi},t}$ constructed by the μ-PolicyChooser are $\frac{\varepsilon}{T}$ accurate, with probability of error is less than δ.

Let π be the input policy at update t to the μ-PolicyChooser. By construction, of h_t, it follows that $\hat{Q}_{\pi,t}(\mu, h_t) \geq \hat{Q}_{\pi,t}(\mu, h)$ for $h \in \Pi_1$. Using, our accuracy condition, this implies that

$$Q_{\pi,t}(\mu, h) - Q_{\pi,t}(\mu, h_t) \leq \frac{2\varepsilon}{T}$$

for either the infinite or finite Π_1 case.

Now the remainder of the proof is similar to that of the exact theorem (6.3.2). Lemma 5.3.1 shows that the state-action values of the output policy $\tilde{\pi}$ are identical to this input policy π, so for all $h \in \Pi_1$

$$Q_{\tilde{\pi},t}(\mu, h) - Q_{\tilde{\pi},t}(\mu, h_t) \leq \frac{2\varepsilon}{T}$$

Hence, $A_{\tilde{\pi},t}(\mu, h) = Q_{\tilde{\pi},t}(\mu, h) - Q_{\tilde{\pi},t}(\mu, h_t)$ and the result follows by setting $\varepsilon \leftarrow \frac{\varepsilon}{2}$. □

6.4. Remarks

6.4.1. The μ-PolicyChooser and Cost/Reward Sensitive Classification. Let examine connections between the loss function used by μ-PolicyChooser and the loss functions in a classification setting. μ-PolicyChooser desires to maximize the function

$$Q_{\pi,t}(\mu, h) = AE_{s\sim\mu(\cdot|t)}E_{a\sim\text{Uniform}}\left[Q_{\pi,t}(s,a)I(h(s)=a)\right]$$

with respect to $h \in \Pi_1$.

Consider the binary classification case where $P(x,y)$ is the joint distribution over input/output pairs, where $y \in \{0,1\}$. The most common loss function for a hypothesis h is $E_{x,y\sim P}\left[I(h(x)=y)\right]$. A weighted loss function variant is

$$E_{x,y\sim P}\left[w(x,y)I(h(x)=y)\right]$$

where $w(x,y)$ are the costs.

For comparison purposes, let us consider a binary action MDP. The input space is the state space and the output space is the binary action space. Each $h \in \Pi_1$ is a "classifier" of actions. The function $-Q_{\pi,t}(\mu,h)$ is just a cost sensitive classification loss function for hypothesis h. The joint distribution P is analogous to the distribution $\mu(\cdot|t) \times$ Uniform and the weights $w(x,y)$ are analogous to the state action values $Q_{\pi,t}(s,a)$ (which are bounded in $[0,1]$).

6.4.2. Optimization and PEGASUS. As in many supervised learning theory analyses, we have made a distinction between the sample and computational complexity and have only addressed the sample complexity. The computational problem of finding a decision rule in $\arg\max_{h\in\Pi_1}\hat{Q}_{\pi,t}(\mu,h)$ still remains. However, as pointed out in the last subsection, this optimization is equivalent to that of optimizing a cost sensitive classification loss function, which is a relatively common problem.

Let us recall the PEGASUS method of Ng and Jordan [2001], where we implement a deterministic generative model on a computer (by "seeding" our random number generator). Here, the only relevant question is one of the computational complexity (see section 6.1.3). In the μ-PolicySearch setting, we have effectively bounded the amount of computation that the deterministic generative model must perform, but the computational cost of the $\arg\max$ optimization remains.

Furthermore, with access to the deterministic generative model of PEGASUS, we might be able to preform some additional tricks to reduce variance. Currently, the μ-PolicyChooser uses importance sampling with $a \sim$ Uniform. Under the deterministic generative model, it might be sensible to try all actions for each state $s \sim \mu(\cdot|t)$ and avoid this importance sampling.

6.4.3. μ-PolicySearch vs. Gradient Methods. This subsection provides an informal comparison of μ-PolicySearch vs. gradient methods. As was done in subsection 5.4.2, we can write the T-epoch gradient as

$$\nabla V_\pi(s_0) = ATE_{(s,t)\sim d_{\pi,s_0}}E_{a\sim\text{Uniform}}\left[A_{\pi,t}(s,a)\nabla\pi(a|s,t,\theta)\right] .$$

In practice, the termination condition of a gradient method is when the magnitude of the gradient is small. As discussed in chapter 4, this is typically when estimation of the gradient *direction* is difficult.

A common termination condition for gradient methods is when

$$||E_{(s,t)\sim d_{\pi,s_0}} E_{a\sim \text{Uniform}} [A_{\pi,t}(s,a)\nabla\pi(a|s,t,\theta)]\,||_2 \leq \varepsilon$$

where ε is "small". Here, $||x||_2$ is the standard l_2 norm ("mean-squared norm"). The guarantee of μ-PolicySearch is that for all $h \in \Pi_1$ and for all t,

$$E_{s\sim\mu(\cdot|t)} E_{a\sim \text{Uniform}} [A_{\pi,t}(s,a) I(h(s) = a)] \leq \varepsilon$$

where ε is "small".

There are two important differences. The first is that μ-PolicySearch guarantees the advantages to be small under the state distribution μ rather than the on-policy distribution d_{π,s_0}. This incorporates the "exploration" into μ-PolicySearch. Through an appropriate choice of μ we have the option of forcing the advantages to be small where we desire. A reasonable heuristic to consider for gradient methods is to use a starting distribution $\tilde{\mu}$ rather than s_0.

The second, and equally important, is that μ-PolicySearch guarantees the advantages to be small with respect to *all* $h \in \Pi_1$ and Π_1 is a potentially *infinite* policy class. In contrast, gradient methods only guarantee the advantages are small with respect to the direction $\nabla\pi(a|s,t,\theta)$. It is hard to understand the implications of this condition for gradient methods. This point is essentially about the efficient use of samples.

Both of these distinctions allow us to make a nontrivial statement on the quality of the output π of μ-PolicySearch (see theorem 6.3.1).

CHAPTER 7

Conservative Policy Iteration

The algorithms we discussed in the last two chapters have required the use of the *non-stationary* policies in order to maximize T-step future reward. These algorithms can also be applied to find a good non-stationary policy in the discounted case, by choosing an appropriate horizon time of $T = O(\frac{-\log \varepsilon}{1-\gamma})$ (see subsection 2.3.3). In many reinforcement learning applications, *stationary* policies are commonly used. This chapter considers the problem of finding a good stationary policy in the γ-discounted case.

It turns out that finding a good stationary policy is considerably harder in an approximate setting. The fundamental difficulty with this case is that an approximate greedy update can have dire consequences, as shown by the max norm bound 3.1.1. The underlying problem with stationary greedy updates is that replacing the old policy at all timesteps allows the worst case error to propagate over the entire horizon. In the non-stationary case, a greedy update only alters the policy at one timestep, and max norm bounds can be avoided (as shown in the previous two chapters).

This chapter introduces the Conservative Policy Iteration (CPI) algorithm. CPI uses *stochastic* policies as a way to avoid making drastic policy changes to a stationary policy. After each policy update in CPI, the new policy is a mixture distribution of the previous policy and a greedy policy (which is returned by a PolicyChooser algorithm).

Recall that μ-PolicySearch guarantees the return of deterministic, non-stationary policy. In contrast, CPI can guarantee the return of a stochastic, stationary policy. Neither algorithm guarantees the return of a policy which is *both* deterministic and stationary. In the Discussion chapter of this thesis, we return to this point.

As in μ-PolicySearch, CPI optimizes a performance criterion that is defined with respect to the measure μ. Again, the use of this measure is a surrogate for explicit exploration. However, in CPI, the μ-reset model used is *stationary*, *ie* μ is only a distribution over states and not times (unlike for μ-PolicySearch). An important distinction between CPI and μ-PolicySearch is that at each policy update in CPI, the algorithm considers improving the policy over its entire horizon, whereas in μ-PolicySearch only one decision rule at a particular timestep is altered.

7.1. Preliminaries

Throughout this chapter, we assume access to a stationary μ-reset model. A **stationary μ-reset model** is a μ-reset model in which μ is stationary, *ie* μ is not time dependent. In the μ-reset model, when given an input time t, the next state was reset to a state $s \sim \mu(\cdot|t)$. In the stationary μ-reset model, the algorithm allows only a reset to $s \sim \mu(\cdot)$ (and no time input is required since μ is not time dependent). Additionally, this model allows the usual

online simulation of MDP. Since we are only dealing with the stationary μ-reset model in this chapter, we just say μ-reset model and the stationarity is clear from context.

We overload value functions in the same manner as was done in the last chapter.

DEFINITION 7.1.1. Let M be an infinite horizon MDP, π be a stationary policy for M, γ be a discount factor, and μ be a distribution over the state space.

The **value** $V_{\pi,\gamma}(\mu)$ of distribution μ is:

$$V_{\pi,\gamma}(\mu) \equiv E_{s\sim\mu}\left[V_{\pi,\gamma}(s)\right] .$$

The **state-action value** $Q_{\pi,\gamma}(\mu, h)$ of a stationary policy h and μ is:

$$Q_{\pi,\gamma}(\mu, h) \equiv E_{s\sim\mu}E_{a\sim h(\cdot|s)}\left[Q_{\pi,\gamma}(s, a)\right] .$$

The **advantage** $A_{\pi,\gamma}(\mu, h)$ of a stationary policy h and μ is:

$$A_{\pi,\gamma}(\mu, h) \equiv E_{s\sim\mu}E_{a\sim h(\cdot|s)}\left[A_{\pi,\gamma}(s, a)\right] .$$

Again, we have the familiar equality:

$$A_{\pi,\gamma}(\mu, h) = Q_{\pi,\gamma}(\mu, h) - V_{\pi,\gamma}(\mu) .$$

We also overload the future state distribution.

DEFINITION 7.1.2. Let M be an MDP, γ be a discount factor, π be a stationary policy, and μ be a distribution over the state space.

$$d_{\pi,\mu,\gamma}(s) \equiv E_{s_0\sim\mu}\left[d_{\pi,s_0,\gamma}(s)\right]$$

Hence, the future state probability $d_{\pi,\gamma,\mu}(s)$ represents the expected future probability of s when the initial state $s_0 \sim \mu$.

This chapter only deals with the discounted case, so we drop the γ-subscripts.

7.2. A Conservative Update Rule

Although, ultimately we may desire a good policy from some distinguished start state s_0, directly improving $V_\pi(s_0)$ has the problem that this measure is not sensitive to improvement at states that are infrequently visited under π (which leads to the "variance trap" of gradient methods, see section 4.3). We return to this point in the Remarks section of this chapter.

Let us consider using the performance measure $V_{\pi,\gamma}(\mu)$ to optimize a policy with respect to the distribution μ. Since we are in the stationary setting, we wish to avoid making greedy updates to a new policy π', since this has the potential for serious policy degradation. Instead, consider improving this measure using the more conservative update rule

$$\pi_{\text{new}}(a|s, \alpha) = (1 - \alpha)\pi(a|s) + \alpha\pi'(a|s) \tag{7.2.1}$$

for some π' and some $\alpha \in [0, 1]$.

This section focuses on understanding improvement with this conservative update rule. Obviously, other update rules are possible, but this rule is particularly simple and leads to sensible improvement guarantees. The following subsections presents these improvement guarantees — the first subsection provides a "small" α improvement condition and the following subsection provides a condition on how large we can safely set α.

7.2.1. The Future Advantage and Policy Improvement. Now let us consider the conditions for policy improvement with respect to α. The gradient (from subsection 5.4.2) with respect to α is

$$\begin{aligned}\frac{\partial}{\partial\alpha}V_{\pi_{\text{new}}}(\mu)|_{\alpha=0} &= \frac{1}{1-\gamma}E_{s\sim d_{\pi,\mu}}\left[\sum_a(\pi'(a|s)-\pi(a|s))A_\pi(s,a)\right]\\ &= \frac{1}{1-\gamma}E_{s\sim d_{\pi,\mu}}E_{a\sim\pi'(\cdot|s)}\left[A_\pi(s,a)\right]\ .\end{aligned}$$

where we have used the fact that $\sum_a \pi(a|s)A_\pi(s,a) = 0$. Importantly, note that the expectation over the state space is with respect to the measure induced by π ($d_{\pi,\mu}$), but the expectation over the action space is with respect to π'.

This motivates the following definitions.

DEFINITION 7.2.1. Let M be an infinite horizon MDP, γ be a discount factor, π be a stationary policy, and μ be a distribution over the state space.

The **future value** $\mathbb{V}_{\pi,\gamma}(\mu)$ of distribution μ is:

$$\mathbb{V}_{\pi,\gamma}(\mu) \equiv E_{s\sim d_{\pi,\mu}}\left[V_{\pi,}(s)\right]\ .$$

The **future state-action value** $\mathbb{Q}_{\pi,\gamma}(\mu,h)$ of a stationary policy h and μ is:

$$\mathbb{Q}_{\pi,\gamma}(\mu,h) \equiv E_{s\sim d_{\pi,\mu}}E_{a\sim h(\cdot|s)}\left[Q_\pi(s,a)\right]\ .$$

The **future advantage** $\mathbb{A}_{\pi,\gamma}(\mu,h)$ of a policy h is:

$$\mathbb{A}_{\pi,\gamma}(\mu,h) \equiv E_{s\sim d_{\pi,\mu}}E_{a\sim h(\cdot|s)}\left[A_\pi(s,a)\right]\ .$$

We drop the γ subscripts when clear from context. Note that the future advantage satisfies

$$\mathbb{A}_\pi(\mu,h) = A_\pi(d_{\pi,\mu},h)$$

and so the future advantage measures the degree to which h is choosing better actions than π with respect to states sampled according to the *future* state distribution $d_{\pi,\mu}$. In Kakade and Langford [2002], this future advantage was referred to as the policy advantage.

As shown above,

$$\frac{\partial}{\partial\alpha}V_{\pi_{\text{new}}}(\mu)|_{\alpha=0} = \frac{1}{1-\gamma}\mathbb{A}_\pi(\mu,\pi')\ .$$

Hence, a sufficiently "small" α can improve the policy if $\mathbb{A}_\pi(\mu,\pi') > 0$. This suggests that we desire a policy π' which chooses actions with large advantages with respect to the future state distribution $d_{\pi,\mu}$.

Contrast this to μ-PolicySearch, where at each step t the goal is to find a decision rule that chooses large advantages with respect to the state distribution $\mu(\cdot|t)$. Here, we desire to find a π' that is "good" with respect to the *entire* horizon (induced by γ).

7.2.2. Non-Trivial Policy Improvement. This "small" α condition is not a powerful enough result to allow us to understand the sample complexity of finding a good policy. We care about how much policy improvement is possible per update and how large we can safely set α during an update. The following theorem provides a bound on how much improvement is possible.

LEMMA 7.2.2. *Let π and π' be stationary policies for an infinite horizon MDP M. Let $\epsilon_\infty = \max_s |E_{a\sim\pi'(\cdot|s)}[A_\pi(s,a)]|$. For the update rule 7.2.1 for all $\alpha \in [0,1]$*

$$V_{\pi_{\text{new}}}(\mu) - V_\pi(\mu) \quad \geq \quad \frac{\alpha}{1-\gamma}\left(\mathbb{A}_\pi(\mu,\pi') - \frac{2\gamma\alpha}{1-\gamma(1-\alpha)}\epsilon_\infty\right).$$

Note that for small α the bound behaves as

$$V_{\pi_{\text{new}}}(\mu) - V_\pi(\mu) \quad \geq \quad \frac{\alpha}{1-\gamma}\mathbb{A}_\pi(\mu,\pi') + O(\alpha^2)$$

which is consistent with the gradient $\frac{\partial}{\partial\alpha}V_{\pi_{\text{new}}}(\mu)|_{\alpha=0}$. Hence, the bound is tight for "small" α for all policies π and π'.

For $\alpha = 1$, the bound reduces to

$$V_{\pi_{\text{new}}}(\mu) - V_\pi(\mu) \quad \geq \quad \frac{\mathbb{A}_\pi(\mu,\pi')}{1-\gamma} - \frac{2\gamma\epsilon_\infty}{1-\gamma}.$$

Note that this penalty term of $\frac{2\gamma\epsilon_\infty}{1-\gamma}$ is analogous to the penalty term in the greedy update bound for function approximation (theorem 3.1.1). There ε was the max norm error in approximating the optimal value function. Consider the case where π is an optimal policy. Then $\varepsilon_\infty = \min_{s,a} A_\pi(s,a)$, since all advantages are non-positive. For this case, ε_∞ is the analogous max norm error made by π'. An example provided in the Remarks section of this chapter shows that this bound is tight for all α and γ for a particular (nontrivial) choice of π' and π.

The intuition for the proof is as follows. The mixing parameter α determines the probability of choosing an action from π'. If the state distribution is $d_{\pi,\mu}$ when an action from π' is chosen, then the performance change is proportional to the future advantage $\mathbb{A}_\pi(\mu,\pi')$. This effect leads to the $O(\alpha)$ first term. However, as α is increased the future state distribution changes so these deviant actions are chosen when the state distribution is not quite $d_{\pi,\mu}$. This latter effects leads to the $O(\alpha^2)$ penalty term.

PROOF. We overload notation by writing $A_\pi(s,h) = E_{a\sim h(\cdot|s)}A_\pi(s,a)$ for a policy h. For any state s,

$$\begin{aligned} A_\pi(s,\pi_{\text{new}}) &= \sum_a \left((1-\alpha)\pi(a|s) + \alpha\pi'(a|s)\right) A_\pi(s,a) \\ &= \alpha\sum_a \pi'(a|s)A_\pi(s,a) \\ &= \alpha A_\pi(s,\pi') \end{aligned}$$

where we have used $\sum_a \pi(a|s)A_\pi(s,a) = 0$.

While following π_{new}, the probability that an action is chosen according to π' is α. Let c_t be a random variable that is 0 if all actions before time t were chosen according to π, and $c_t = 1$ else. Hence, $\Pr(c_t = 0) = (1-\alpha)^t$, and define $\rho_t \equiv \Pr(c_t = 1) = 1-(1-\alpha)^t$. As usual, $\Pr(s_t = s|h,M,\mu)$ is the probability that the state at time t is s while following

the policy h starting from $s_0 \sim \mu$. The μ and M dependence are clear from context, so their dependence is suppressed. By definition of c_t, we have

$$\Pr(s_t = s|\pi_{\text{new}}, c_t = 0) = \Pr(s_t = s|\pi)\,.$$

Hence,

$$\begin{aligned}
& E_{s\sim\Pr(s_t=s|\pi_{\text{new}})}\left[A_\pi(s,\pi_{\text{new}})\right] \\
= \;& \alpha E_{s\sim\Pr(s_t=s|\pi_{\text{new}})}\left[A_\pi(s,\pi')\right] \\
= \;& \alpha(1-\rho_t)E_{s\sim\Pr(s_t=s|\pi_{\text{new}},c_t=0)}\left[A_\pi(s,\pi')\right] + \alpha\rho_t E_{s\sim\Pr(s_t=s|\pi_{\text{new}},c_t=1)}\left[A_\pi(s,\pi')\right] \\
\geq \;& \alpha(1-\rho_t)E_{s\sim\Pr(s_t=s|\pi)}\left[A_\pi(s,\pi')\right] - \alpha\rho_t\varepsilon_\infty \\
\geq \;& \alpha E_{s\sim\Pr(s_t=s|\pi)}\left[A_\pi(s,\pi')\right] - 2\alpha\rho_t\varepsilon_\infty
\end{aligned}$$

where we have used the definition of ε_∞.

Using the performance difference lemma (5.2.1), we have

$$\begin{aligned}
V_{\pi_{\text{new}}}(\mu) - V_\pi(\mu) &= \frac{1}{1-\gamma}E_{s\sim d_{\pi_{\text{new}},\mu}}E_{a\sim\pi_{\text{new}}(\cdot|s)}\left[A_\pi(s,a)\right] \\
&= \sum_t \gamma^t E_{s\sim P(s_t=s|\pi_{\text{new}})}\left[A_\pi(s,\pi_{\text{new}})\right] \\
&\geq \sum_t \gamma^t\left(\alpha E_{s\sim P(s_t=s|\pi)}\left[A_\pi(s,\pi')\right] - 2\alpha\rho_t\varepsilon_\infty\right) \\
&= \frac{\alpha}{1-\gamma}E_{s\sim d_{\pi,\mu}}\left[A_\pi(s,\pi')\right] - 2\alpha\varepsilon_\infty\sum_t\gamma^t(1-(1-\alpha)^t) \\
&= \frac{\alpha}{1-\gamma}\mathbb{A}_\pi(\mu,\pi') - 2\alpha\varepsilon_\infty\left(\frac{1}{1-\gamma} - \frac{1}{1-\gamma(1-\alpha)}\right) \\
&= \frac{\alpha}{1-\gamma}\left(\mathbb{A}_\pi(\mu,\pi') - \frac{2\gamma\alpha}{1-\gamma(1-\alpha)}\varepsilon_\infty\right)
\end{aligned}$$

which proves the lemma. □

Provided that $\mathbb{A}_\pi(\mu,\pi')$ is greater than 0, then a value of α can be determined to guarantee a certain amount of improvement.

COROLLARY 7.2.3. *Assume the setting in theorem 7.2.2. If $\mathbb{A}_\pi(\mu,\pi') \geq 0$, then setting*

$$\alpha = \frac{1-\gamma}{4}\mathbb{A}_\pi(\mu,\pi')$$

leads to the following policy improvement

$$V_{\pi_{new}}(\mu) - V_\pi(\mu) \quad \geq \quad \mathbb{A}_\pi(\mu,\pi')^2/8\,.$$

PROOF. Since $\varepsilon_\infty \leq 1$ and $0 \leq \alpha \leq 1$, it follows that

$$V_{\pi_{\text{new}}}(\mu) - V_\pi(\mu) \geq \frac{\alpha}{1-\gamma}\mathbb{A}_\pi(\mu,\pi') - \frac{2\alpha^2}{(1-\gamma)^2}$$

Optimizing this quadratic expression with respect to α leads to the result. □

This corollary suggests choosing policies π' with large future advantages.

7.3. Conservative Policy Iteration

As in the μ-PolicySearch case, the goal of the algorithm is to return a policy that has small advantages on average, except we now desire a stationary policy. In order to make the comparison to the T-step case more evident, define the horizon time H as

$$H \equiv \frac{1}{1-\gamma}.$$

This definition was also used for comparisons in section 2.5.

Recall from the performance difference lemma (5.2.1) that the difference in value between an optimal policy π^* and π at s_0 is

$$V_{\pi^*}(s_0) - V_\pi(s_0) = H E_{s\sim d_{\pi^*,s_0}} E_{a\sim\pi^*(\cdot|s)}\left[A_\pi(s,a)\right] .$$

Therefore, if each advantage is less than ε/H then π has value ε near to the optimal value. Instead of attempting to ensure that each advantage is small (which is a max norm condition), CPI attempts to ensure that on average the advantages are ε/H small.

We consider a "policy search" setting for CPI, where the algorithm uses a restricted class of stationary policies Π (see Kakade and Langford [2002] for a slightly different version of CPI). The goal of CPI is to find a policy π such that for all $h \in \Pi$

$$\mathbb{A}_\pi(\mu, h) \leq \frac{\varepsilon}{H} .$$

This guarantee is similar to that of μ-PolicySearch, except now the future advantage is the natural quantity to consider (as opposed to the advantage $A_\pi(\mu, h)$).

The first subsection discusses optimality guarantees, the next subsection presents the main results of CPI, and the final subsection provides the sample complexity analysis. Note that we expect the analysis to be somewhat more involved than that of μ-PolicySearch, since μ-PolicySearch naturally terminates after T calls to the PolicyChooser. However, for our conservative update rule, we must explicitly decide when to terminate policy updates in order to obtain finite sample size bounds. The performance bound from the previous section helps us here.

7.3.1. μ-Optimality. This subsection examines the analogous performance guarantees to that of μ-PolicySearch. Note that if for all $h \in \Pi$, $\mathbb{A}_\pi(\mu, h) \leq \frac{\varepsilon}{H}$, then

$$\begin{aligned} A_\pi(d_{\pi,\mu}, h) &\leq \frac{\varepsilon}{H} \\ A_\pi(\mu, h) &\leq \varepsilon . \end{aligned}$$

The first statement is due to the fact that $\mathbb{A}_\pi(\mu, h) = A_\pi(d_{\pi,\mu}, h)$. The second statement is due to the fact that $d_{\pi,\mu} \geq \frac{\mu}{H}$, which follows from the definition of future advantage (the future distribution has a $\frac{\mu}{H}$ contribution form the starting distribution). This implies that $\mathbb{A}_\pi(\mu, h) \geq \frac{1}{H} A_\pi(\mu, h)$, and so $A_\pi(\mu, h) \leq \varepsilon$.

Note that that we effectively have a "local" and "non-local" guarantee. The local guarantee is that π has small advantages with respect to the states that it currently visits (*ie* with respect to $d_{\pi,\mu}$). The "non-local" guarantee is that π has small advantages with respect to μ. However, note that this μ condition is a factor of H worse (the bound is ε rather than $\frac{\varepsilon}{H}$).

The following lemma gives reassurance as to why this is a sensible goal. Again, define $||p - q||_1 = \sum_{x \in \mathcal{X}} |p(x) - q(x)|$, and if $\mathcal{X}$ is a continuous space, the sum is replaced by an integral. Importantly, note that the theorem is stated assuming a bound of $\frac{\varepsilon}{H}$ on the advantage A_π and not the future advantage $\mathbb{A}_\pi$ (see the equations above).

THEOREM 7.3.1. *Let Π be a class of stationary policies for an infinite horizon MDP M. Assume that π is a policy such that for the distribution ν and for all $h \in \Pi$,*

$$A_\pi(\nu, h) \leq \frac{\varepsilon}{H}$$

Then for policies $\pi' \in \Pi$ and for all s_0,

$$V_\pi(s_0) \geq V_{\pi'}(s_0) - \varepsilon - H||d_{\pi', s_0} - \nu||_1 \,.$$

So if $\nu = d_{\pi,\mu}$, the policy π is guaranteed to compete favorably against any policy $\pi' \in \Pi$ whose future state distribution is close to $d_{\pi,\mu}$. If $A_\pi(\mu, h) \leq \varepsilon$, then we also have the weaker guarantee (by a factor of H) that π competes favorably against those $\pi' \in \Pi$ whose future state distribution is close to μ.

Essentially, the proof uses the notion that we are being "tested" under a different distribution than our "training" distribution and is similar to the proof of the analogous theorem for the T-case (6.3.1).

PROOF. Using the performance difference lemma (5.2.1) and the fact that $\pi' \in \Pi$,

$$\begin{aligned} V_{\pi'}(s_0) - V_\pi(s_0) &= H E_{s \sim d_{\pi', s_0}} E_{a \sim \pi'(\cdot|s)} \left[A_\pi(s, a)\right] \\ &\leq H E_{s \sim \nu} E_{a \sim \pi'(\cdot|s)} \left[A_\pi(s, a)\right] + H||d_{\pi', s_0} - \nu||_1 \\ &= H A_\pi(\nu, \pi') + H||d_{\pi', s_0} - \nu||_1 \\ &\leq \varepsilon + H||d_{\pi', s_0} - \nu||_1 \end{aligned}$$

where the last step follows by assumption. □

7.3.2. The Algorithm. The high level idea of CPI is straightforward. The algorithm calls the PolicyChooser to obtain a π' and then performs a conservative update with this π'. The immediate questions are: when should the algorithm terminate and how should the PolicyChooser be implemented? Let us first specify an exact version of CPI, which gives insight into the sample based version

In Exact μ-PolicySearch (algorithm 11), the algorithm chooses the decision rule $h \in \Pi_1$ to maximize $Q_{\pi,t}(\mu, h)$. In this setting, it is the future values that are relevant, so the exact algorithm chooses $h \in \Pi$ to maximize $\mathbb{Q}_\pi(\mu, h)$.

We now specify a termination condition. Recall that if the PolicyChooser algorithm ever returns a policy π' where $\mathbb{A}_\pi(\mu, \pi') \leq 0$ then improvement is no longer guaranteed. The termination condition used by CPI is that it continues to update the policy as long as the PolicyChooser returns a π' such that $\mathbb{A}_\pi(\mu, \pi') \geq \frac{\varepsilon}{H}$. Note this condition translates into a small gradient condition, since

$$\frac{\partial}{\partial \alpha} V_{\pi_{\text{new}}}(\mu)|_{\alpha=0} = H \mathbb{A}_\pi(\mu, \pi') \leq \varepsilon$$

However, this "gradient" is determined by the π' that is returned by the PolicyChooser, which chooses π' from a potentially infinite set Π (see the Remarks section in the last chapter for a comparison of these methods to gradient methods).

The exact version of CPI is shown in algorithm 11. In addition to assuming that an exact PolicyChooser is used, the algorithm assumes that it can set the value of α using the exact future advantage $\mathbb{A}_\pi(\mu, \pi')$. The algorithm then sets α to the value specified in corollary 7.2.3 in order to improve $V_\pi(\mu)$ by $O(\mathbb{A}_\pi(\mu, \pi')^2)$. Both of these assumptions are removed in the following subsections when we consider the a sample based version of CPI.

Algorithm 11 Exact CPI

(1) Randomly initialize π
(2) Call the ExactPolicyChooser:

$$\pi' = \arg\max_{h \in \Pi} \mathbb{Q}_\pi(\mu, h)$$

(3) If $\mathbb{A}_\pi(\mu, \pi') > \frac{\varepsilon}{H}$,
  (a) set

$$\alpha = \frac{\mathbb{A}_\pi(\mu, \pi')}{4H}$$

  (b) perform the update:

$$\pi \leftarrow (1 - \alpha)\pi + \alpha\pi'$$

  (c) go to step 2.
(4) Else, HALT and return π

7.3.3. The Main Results. The following theorem is on the guarantee of this exact algorithm.

THEOREM 7.3.2. *(Exact CPI) Let Π be a class of stationary policies for an infinite horizon MDP M. Exact CPI improves the value $V_\pi(\mu)$ after each update and halts after $O(\frac{H^2}{\varepsilon^2})$ calls to the ExactPolicyChooser (in line 2 of the algorithm). Furthermore, the policy π returned by Exact CPI is stationary and satisfies, for all $h \in \Pi$,*

$$\mathbb{A}_\pi(\mu, h) \leq \frac{\varepsilon}{H}.$$

PROOF. At every policy update $\mathbb{A}_\pi(\mu, h) \geq \frac{\varepsilon}{H}$, so corollary 7.2.3 implies that $V_\pi(\mu)$ improves by $O(\frac{\varepsilon^2}{H^2})$ after each update. Since this value function is bounded by 1, the algorithm must halt in $O(\frac{H^2}{\varepsilon^2})$. After the algorithm halts, the algorithm must have obtained some π' such that $\mathbb{A}_\pi(\mu, \pi') \leq \frac{\varepsilon}{H}$. By construction, for this π' and for all $h \in \Pi$,

$$\mathbb{Q}_\pi(\mu, h) \leq \mathbb{Q}_\pi(\mu, \pi').$$

Since $\mathbb{A}_\pi(\mu, h) = \mathbb{Q}_\pi(\mu, h) - \mathbb{V}_\pi(\mu)$ then for all $h \in \Pi$,

$$\mathbb{A}_\pi(\mu, h) \leq \mathbb{A}_\pi(\mu, \pi') \leq \frac{\varepsilon}{H}$$

which completes the proof. □

Interestingly, note that the policy π returned is *not* necessarily in Π. In fact, Π could be a *deterministic* policy class and yet the output policy π is *stochastic*, in general.

Obviously, the exact algorithm is impractical. The following theorem is on the sample complexity of a sample based CPI algorithm which assumes access to the μ-reset model. This algorithm is spelled out the in following subsections.

THEOREM 7.3.3. *(Sample Based CPI) Let* Π *be a class of stationary deterministic policies for an infinite horizon MDP* M*. For any* ε *and* δ*, with probability greater than* $1-\delta$*, CPI observes a number of transitions that is*

- *if* Π_1 *is finite,*

$$O\left(\frac{A^2H^5\log\varepsilon}{\varepsilon^4}(\log|\Pi_1|+\log\frac{H}{\varepsilon\delta})\right)$$

- *if* Π_1 *is infinite and* M *is a binary action MDP,*

$$O\left(\frac{H^5\log\varepsilon}{\varepsilon^4}(VC(\Pi)+\log\frac{H}{\varepsilon\delta})\right)$$

and, for either case, CPI returns a stationary policy π *such that for all* $h\in\Pi$

$$\mathbb{A}_\pi(\mu,h)\leq\frac{\varepsilon}{H}.$$

Again, we have a *polynomial* T dependence, *no dependence* on the size of the state space, and a dependence on the complexity of Π that is comparable to the trajectory tree method. In comparison to the sample complexity bounds for the μ-PolicySearch algorithm (theorem 6.3.3), the polynomial factors of H and ε are slightly higher.

The next subsection discusses the construction of the sample based PolicyChooser and the following section completes the proof.

7.3.4. The "Stationary" μ-PolicyChooser. As in subsections 6.3.3 and 6.4.1, we can write our cost function as a "cost sensitive classification loss"

$$\mathbb{Q}_\pi(\mu,h)=AE_{s\sim d_{\pi,\mu}}E_{a\sim\text{Uniform}}\left[Q_\pi(s,a)I(h(s)=a)\right]$$

which implies a sampling procedure. In comparison to the T-step case, the sampling procedure has the additional complication that our horizon time is infinite. First, the sampling procedure is outlined assuming that we can obtain samples from $d_{\pi,\mu}$ and unbiased estimates of $Q_\pi(s,a)$. This procedure is similar to the one of μ-PolicySearch (see subsection 6.3.3). First, obtain an $s\sim d_{\pi,\mu}$ and an $a\sim$ Uniform (see section 4.2.1 for this sampling procedure), and then obtain an unbiased estimate $\hat{Q}$ of $Q_\pi(s,a)$. Then $AI(h(s)=a)\hat{Q}$ is an unbiased estimate of $\mathbb{Q}_\pi(\mu,h)$. Let $\hat{Q}_i$ be the i-th sample with a corresponding state-action of (s_i,a_i) . If m samples are obtained, then an estimate of $\mathbb{Q}_\pi(\mu,h)$ is

$$\widehat{\mathbb{Q}}_\pi(\mu,h)=\frac{A}{m}\sum_i\hat{Q}_iI\left(h(s_i)=a_i\right)$$

for any $h\in\Pi$.

To deal with the infinite horizon, let us impose a horizon of $O(H\log\varepsilon)$ which introduces a bias of ε (see section 2.3.3). The biased sampling procedure from $d_{\pi,\mu}$ is as follows. First, sample $s_0\sim\mu$, then follow the policy π. At each step, accept the current state as a sample with probability $1/H$. If the time $O(H\log\varepsilon)$ is reached, then accept the current state as the sample. A biased sample of $Q_\pi(s,a)$ is the empirical discounted reward (normalized by $1/H$) obtained by simulating π over the horizon time of $O(H\log\varepsilon)$. This procedure introduces a bias of $O(\varepsilon)$ into our estimate $\widehat{\mathbb{Q}}_\pi(\mu,h)$. The total number of transitions observed by μ-PolicyChooser under this method is $O(mH\log\varepsilon)$.

The complete Stationary μ-PolicyChooser is shown in algorithm 12 (we say stationary to distinguish this μ-PolicyChooser from the one in the last chapter). The policy returned is just the policy $\pi'\in\Pi$ that optimizes the empirical estimate $\widehat{\mathbb{Q}}_\pi(\mu,h)$.

Algorithm 12 "Stationary" μ-PolicyChooser(π, H)

(1) Obtain a biased sample set $\{(s_i, a_i, \hat{Q}_i)\}$ of size m as described in the text
(2) Construct the estimates

$$\widehat{\mathbb{Q}}_\pi(\mu, h) = \frac{A}{m} \sum_i \hat{Q}_i I(h(s_i) = a_i)$$

(3) Set

$$\pi' = \arg\max_{h \in \Pi} \widehat{\mathbb{Q}}_\pi(\mu, h)$$

(4) Return π'

The following uniform convergence lemma is useful in establishing the soundness of CPI. This lemma is analogous to the sample size lemmas (6.3.4 and 6.3.5) of the last chapter. Recall that $O(mH \log \varepsilon)$ is the total number of samples observed by the PolicyChooser.

LEMMA 7.3.4. *Let Π be a finite class of stationary policies for an MDP M. Let m be set as follows:*

- *if Π_1 is finite*

$$m = O\left(\frac{A^2 H^2}{\varepsilon^2}(\log|\Pi| + \log\frac{1}{\delta})\right)$$

- *if Π_1 is infinite and M is a binary action MDP*

$$m = O\left(\frac{H^2}{\varepsilon^2}(\log VC(\Pi) + \log\frac{1}{\delta})\right)$$

then, for either of these two cases and upon input of of a policy π, μ-PolicySearch constructs a function $\widehat{\mathbb{Q}}_\pi(\mu, h)$ such that with probability greater than $1 - \delta$, for all $h \in \Pi$

$$|\widehat{\mathbb{Q}}_\pi(\mu, h) - \mathbb{Q}_\pi(\mu, h)| \leq \frac{\varepsilon}{H}.$$

PROOF. The sampling procedure has two forms of bias. One is from biased samples of $d_{\pi,\mu}$ and the other is from using a cutoff value function $(1-\gamma)E\left[\sum_{\tau=0}^{T'} \gamma^\tau r(s_\tau, a_\tau)\right]$ where T' is the cutoff. With a horizon time of $T' = O(H \log \varepsilon)$, then the bias in this function can be reduced to $O(\varepsilon)$. Also using this T', the cutoff sampling distribution for $d_{\pi,\mu}$ introduces another bias term of $O(\varepsilon)$ into the expectations taken. It suffices to show that the biased estimates are $\frac{\varepsilon}{2}$ accurate with probability of error less than δ. The remainder of the proof is identical to the proofs of lemmas 6.3.4 and 6.3.5 except now the analysis is applied to these biased functions. □

7.3.5. Completing the proof. We are almost ready to specify the algorithm and proof. There is the remaining technicality that given an output policy π', we wish to estimate $\mathbb{A}_\pi(\mu, \pi')$ in order to set α. Since $\mathbb{A}_\pi(\mu, \pi') = \mathbb{Q}_\pi(\mu, \pi') - \mathbb{Q}_\pi(\mu, \pi)$, we can estimate the future advantages as follows:

$$\widehat{\mathbb{A}}_\pi(\mu, \pi') = \widehat{\mathbb{Q}}_\pi(\mu, \pi') - \widehat{\mathbb{Q}}_\pi(\mu, \pi)$$

The previous lemma ensures us that our estimate of $\widehat{\mathbb{Q}}_\pi(\mu, \pi')$ is accurate. Additionally, we just need to ensure accuracy on $\widehat{\mathbb{Q}}_\pi(\mu, \pi)$, (since π is not necessarily in Π, the previous lemma does not guarantee accuracy on this input). However, it easy to demand that $\widehat{\mathbb{Q}}_\pi(\mu, \pi)$ also be accurate.

Algorithm 13 CPI(μ-reset,Π)

(1) Randomly initialize π
(2) Obtain π'
$$\pi' = \mu \text{ PolicyChooser}(\pi, \Pi)$$
(3) If $\widehat{\mathbb{A}}_\pi(\mu, \pi') > \frac{2\varepsilon}{3H}$, then
  (a) set
$$\alpha = \frac{\widehat{\mathbb{A}}_\pi(\mu, \pi') - \frac{\varepsilon}{3H}}{4H}$$
  (b) perform the update
$$\pi \leftarrow (1-\alpha)\pi + \alpha\pi'$$
  (c) go to 2.
(4) Else, HALT and return π

The CPI algorithm is presented in algorithm 13. The value of α is now set using $\widehat{\mathbb{A}}_\pi(\mu, \pi') - \frac{\varepsilon}{3H}$. The factor of $\frac{\varepsilon}{3H}$ is due to the technicality that we are using estimates of $\mathbb{A}_\pi(\mu, \pi)$, so this accounts for the approximation error (see the following proof).

The proof of theorem 7.3.3 follows.

PROOF. (proof of 7.3.3) For now, assume that estimates
$$\widehat{\mathbb{A}}_\pi(\mu, h) = \widehat{\mathbb{Q}}_\pi(\mu, h) - \widehat{\mathbb{Q}}_\pi(\mu, \pi)$$
are $\frac{\varepsilon}{3H}$ accurate for all $h \in \Pi$ at every step of the algorithm. Then at every update the true future advantage of π' satisfies $\mathbb{A}_\pi(\mu, \pi') > \frac{\varepsilon}{3H}$ (by line 4 in the algorithm). By corollary 7.2.3, the improvement of $V_\pi(\mu)$ must be $O(\frac{\varepsilon^2}{H^2})$, and so the algorithm must halt in $O(\frac{H^2}{\varepsilon^2})$ steps. Furthermore, by step 4, the algorithm halts when $\widehat{\mathbb{A}}_\pi(\mu, \pi') \leq \frac{2\varepsilon}{3H}$. By construction (and the above equation), the policy π' returned by the μ-PolicyChooser is in $\arg\max_{h \in \Pi} \widehat{\mathbb{A}}_\pi(\mu, h)$. Therefore, when the algorithm ceases, the policy π returned satisfies, for all $h \in \Pi$, $\mathbb{A}_\pi(\mu, h) \leq \varepsilon$.

The remainder of proof involves determining the sample size such that at every step our estimation error in $\widehat{\mathbb{A}}_\pi(\mu, h)$ is $\frac{\varepsilon}{3H}$ for all $h \in \Pi$. This occurs if the estimates of $\widehat{\mathbb{Q}}_\pi(\mu, h)$ and $\widehat{\mathbb{Q}}_\pi(\mu, \pi)$ are $\frac{\varepsilon}{6H}$ accurate. Lemma 7.3.4 provides a value of m in order to obtain $O(\frac{\varepsilon}{H})$ accurate estimates for one call to PolicyChooser with probability of error less than δ. Hence, we must ensure that this accuracy condition is satisfied for all $O(\frac{H^2}{\varepsilon^2})$ calls to PolicyChooser, so replace $\delta \leftarrow \frac{\varepsilon^2\delta}{H^2}$. After this replacement, the proof follows by noting that total number of observed transitions is $O(\frac{mH^3 \log \varepsilon}{\varepsilon^2})$, since each call to the PolicyChooser requires $O(mH \log \varepsilon)$ transitions and there are $O(\frac{H^2}{\varepsilon^2})$ calls to the PolicyChooser (with probability of error less than δ). □

7.4. Remarks

7.4.1. What about improving $V_\pi(s_0)$? In policy gradient methods, we are often interested in improving the policy under some distinguished performance measure $V_\pi(s_0)$, where the state s_0 is some distinguished start state. Here, we argued that even if ultimately, we desire a good policy from s_0, it is important to improve the measure $V_\pi(\mu)$ (where

μ is some distribution that hopefully reflects where good policies from s_0 tend to visit). Furthermore, CPI reliably improves $V_\pi(\mu)$ at every step before the algorithm halts.

A natural question is: can we jointly improve both $V_\pi(s_0)$ and $V_\pi(\mu)$? Under exact methods this is certainly possible. However, under this approximate scheme the general answer is no (unless we expend an extreme number of samples). The reason is again related to those situations discussed in chapter 4, where it is hard to determine the gradient of $V_\pi(s_0)$ due to the problems related to exploration. However, it might be sensible to try and use the measure $\tilde{\mu} = (1-\beta)\mu + \beta\delta_{s_0}$, where δ_{s_0} is the distribution in which $\delta_{s_0}(s_0) = 1$. Essentially, this gives β weight to the distinguished start-state. Here, it might be possible to "often" improve $V_\pi(s_0)$.

7.4.2. Greedy Improvement and Line Searches. The greedy update bound uses a pessimistic value of α, since it sets α based on the worst case bound in lemma 7.2.2. It would certainly be sensible to try one dimensional line searches over α to improve the policy, since in practice it is unlikely we are in this worse case.

However, as the following example shows, the improvement bound 7.2.2 can be tight, simultaneously for all α and γ.

EXAMPLE 7.4.1. There are two states, i and j, and two actions, 1 and 2, at each state. At state i, action 1 is a self transition with $r(i,1) = \frac{1}{2}$ and action 2 transitions to state j for maximal reward $r(i,2) = 1$. At state j, both actions lead to self transitions and $r(j,1) = \frac{1}{2}$ and $r(j,2) = 0$. Let the reset distribution be the state i (*ie* $\mu(i) = 1$).

Consider the starting with the deterministic policy $\pi(i) = 1$ and $\pi(j) = 1$, so $d_{\pi,i}(i) = 1$. The advantages are $A_\pi(i,2) = (1-\gamma)/2$ and $A_\pi(j,2) = -(1-\gamma)/2$ (and the advantages are 0 for the actions taken by π). Consider a policy π' such that $\pi'(i) = 2$ and $\pi'(j) = 2$. The future advantage is then

$$\begin{aligned} \mathbb{A}_\pi(i,\pi') &= 1*(1-\gamma)/2 + 0*(-(1-\gamma)/2) \\ &= (1-\gamma)/2 \end{aligned}$$

and so policy improvement of π is possible using π' since this quantity is positive. Also, $\varepsilon_\infty = (1-\gamma)/2$. Substituting these quantities into the bound gives:

$$V_{\pi_{\text{new}}}(1) - V_\pi(1) \geq \frac{\alpha}{2} - \frac{\gamma\alpha^2}{1-\gamma(1-\alpha)}$$

where $\pi_{\text{new}} = (1-\alpha)\pi + \alpha\pi'$.

It is clear that $V_\pi(1) = \frac{1}{2}$ and using some algebra it can be shown that

$$V_{\pi_{\text{new}}}(1) = \frac{1}{2} + \frac{\alpha}{2} - \frac{\gamma\alpha^2}{1-\gamma(1-\alpha)}$$

which shows that the bound is tight for all α and γ for this π' and π.

Part 3

Exploration

CHAPTER 8

On the Sample Complexity of Exploration

We now turn to the purest reinforcement learning setting in which we are only able to simulate the MDP in an online manner, and we don't have access to either a generative model or μ-reset model. Here, the "exploration/exploitation" dilemma is with us strongly. Without knowledge of the MDP, some of this time will be spent gaining knowledge about the MDP ("exploration") and some time will be spent using this knowledge to obtain reward ("exploitation").

This chapter examines how much time an agent must spend "exploring" in order for it to act near optimally. This question is ill posed and we first suggest one notion for the "sample complexity of exploration" motivated by the E^3 ("Explicit Explore or Exploit") algorithm of of Kearns and Singh [1998] (also see Fiechter [1994] for a precursor to this algorithm).

8.0.3. Notions of Optimality. First, consider the case in which the agent has knowledge that it is placed in an L-epoch MDP. In this chapter, we use L to denote the number of epochs (and reserve T as a planning horizon time used by our algorithm). Often the most sensible goal for the agent is to maximize the total reward over the L decision epochs. Obviously if the agent has knowledge of the MDP, this has a well defined solution. Consider the weaker setting in which the agent only has knowledge that it is in some MDP M that was chosen according to some distribution Q, *ie* at the start $M \sim Q$ and then agent the acts in M for L epochs (and the agent learns about M through the actions it takes). An optimal strategy is one which optimizes the total expected reward in L steps, where the expectation is taken with respect to Q and the strategy. Note that from this point of view the exploration/exploitation tradeoff is artificial, since all actions of an optimal strategy are chosen to maximize the total reward. For a single state MDP, an optimal efficient algorithm exists using Gittins indexes (Gittins [1989]). The multi-state case can be cast as a partially observable Markov Decision Problem whose adverse computational costs are well understood (see Littman [1996]). We return to this point in the discussion of this thesis.

In a more agnostic setting, one might not assume knowledge of Q. Now the notion of optimality is less well defined. One could consider an adversarial setting where one assumes that Nature is malicious. Here, we may desire an algorithm that is competitive against this malicious Nature which picks Q after we choose our algorithm. This analysis has not been formalized in the MDP setting and it is likely to lead to relatively weak guarantees (an idealized setting that is not directly applicable to MDPs was considered in Langford, Zinkevich, and Kakade [2002]).

Also working in this agnostic setting, the E^3 algorithm satisfies a somewhat different and more tractable goal (see Brafman and Tennenholtz [2001] and Kearns and Koller [1999] for subsequent generalizations). The guarantee of the algorithm is different for the discounted

and undiscounted cases, and the guarantee does not explicitly take into account the number of decision epochs L.

We summarize the γ-discounted case first. The question that is addressed is how many transitions does an agent need to observe in the MDP before it arrives at a state such that, with some certainty, the agent has a policy which has value near to the optimal discounted value from that state. The E^3 algorithm can make this claim in time that is polynomial in the size of the state space, action space, $\frac{1}{1-\gamma}$, and relevant factors for the certainty and approximation parameters. By time, we mean both the number of observed transitions in the MDP and off-line computation time.

For the undiscounted, finite T-step horizon case, the algorithm compares itself with policies that "mix" in time T. Roughly speaking, a distribution "mixes" in time T if the distribution is close to its stationary distribution after T timesteps.[1] Here, the guarantee is that the algorithm achieves an average reward that is comparable to the maximal average reward of those policies that "mix" in time T and that this occurs in time polynomial in the relevant quantities. In contrast to the discounted case, this algorithm does not halt, and continues to execute many T-step near-optimal polices.

For our purposes, we find the distinction between the statements for the discounted and undiscounted cases somewhat artificial. Notions of "mixing" seem irrelevant to the behavior of the E^3 algorithm. For the T-step case, one could modify the E^3 algorithm such that it halts and returns a T-step optimal policy from some state, which is a guarantee that is more parsimonious to the γ-discounted case. For this discounted case, halting also seems unnecessary. One could again make a more parsimonious guarantee to the T-step case by modifying the algorithm to compete with those policies that "mix" in time $\frac{1}{1-\gamma}$ (see Baxter and Bartlett [2001] and Kakade [2001] for connections between the mixing time and a discount factor).

8.0.4. The Sample Complexity of Exploration. This chapter takes a different interpretation of these results. In our setting, we assume the agent is in an L-epoch MDP. Additionally, we assume that some planning horizon based on γ or T is imposed (where T is some less than L). Informally, the question of interest is: *For each of the L visited states, does the agent act near-optimally with respect to γ or T from that state?* This notion is formalized by considering the agent to be just an algorithm and then considering the expected reward obtained by the algorithm itself. Recall that the value of the sparse sampling algorithm itself is near-optimal from every state the algorithm visits (see section 2.5). Without knowledge of the MDP (or access to a generative model as the sparse sampling algorithm assumes), then it is unreasonable to expect to an algorithm to act near-optimally from all states the algorithm visits.

Once a horizon based on T or γ has been imposed (which is not equal to L), there is a natural "exploration/exploitation" distinction. As the agent obtains information about the MDP it could alter its behavior based on this experience such that it attempts to act near-optimally (with respect to T or γ) from subsequent states. Loosely, those states (or equivalently timesteps) in the L-epoch horizon in which the agent acts near-optimally can be labelled as "exploitation" and those states (or equivalently timesteps) in which the agent is not acting near optimally can be labelled as "exploration". We view this number of times

[1] See Kearns and Singh [1998] for their slightly weaker notion of mixing, which is stated with respect to the time in which the average T-step return approaches the average infinite horizon return.

in which the agent is not acting near-optimally as the *"sample complexity of exploration"* (and this notion is with respect to T or γ).

Note that this is a rather different question than maximizing the total reward in L steps. However, in a more agnostic setting it is unreasonable for the agent to find an optimal L-epoch policy with only L observed transitions. The E^3's notions of mixing gives us some reassurance as to why planning with respect to a smaller horizon based on T or γ may be a sensible goal. However, the guarantees themselves can be stated more generally without any notions of mixing.

The original E^3 statements were primarily interested in providing only polynomial time guarantees and the actual bounds are somewhat loose. This chapter is focused on placing tight upper and lower bounds on the sample complexity. It turns out that the R_{max} algorithm of Brafman and Tennenholtz [2001] allows us to make stronger guarantees with respect to our aforementioned sample complexity question. The R_{max} algorithm is a generalization of the E^3 algorithm (to stochastic games), which *implicitly* makes the exploration/exploitation tradeoff by directly rewarding uncertainty.

We first formalize our notion of the sample complexity of exploration for both the T-step and γ-discounted case and then present our results. Importantly, the number of wasted steps on exploration is bounded *independently* of L, so regardless of the number of states visited in the MDP, there is only a fixed number of states in which the agent is "exploring". The results for the both the T and γ case are analogous and this chapter focuses on the T-step case for clarity. An upper bound is provided which states that the algorithm must waste $O(\frac{N^2AT^3}{\varepsilon^3}\log^2\frac{NA}{\delta})$ actions on exploration, which is perhaps intuitively appealing, since N^2A is the number of parameters used to specify the transition matrix.

Unfortunately, a lower bound is presented which is only $O(\frac{NAT}{\varepsilon}\log\frac{1}{\delta})$. This lower bound is identical to the lower bound where the agent has access to a generative model (as in the PhasedValueIteration algorithm of section 2.3.2). This lower bound suggests that building an accurate model of the MDP is not required. It is not clear if the gap lies in a loose upper or lower bound. This issue of accurate model building is focused on in the next chapter, where examine this gap more closely.

This chapter also presents results on the case in which the MDP is deterministic, which are related to results in Koenig and Simmons [1993]. Here, our upper and lower bounds are identical and are $O(NAT)$. Papadimitriou and Tsitsiklis [1987] show that the complexity classes are different for general and deterministic MDPs with respect to computing an optimal policy given complete knowledge of the MDP (also see Littman, Dean, and Kaelbling [1995]). In light of this result, it is not unreasonable to obtain different results for the deterministic and stochastic case.

8.1. Preliminaries

Again, we use our standard definitions of an L-epoch MDP, where there are N states and A actions. We use L as the number of epochs and reserve T as the time used by our planning algorithm. It is important to note that in this chapter the agent only has access to the **online simulation model** of the MDP (as defined in section 2.4), which does not allow any "reset" actions. In the online simulation model, the agent is started at state s_0 and follows one unbroken path of experience determined by the transition model of the MDP and the actions chosen by the agent.

The following definition is useful.

DEFINITION 8.1.1. Let M be an L-epoch MDP and let t be a time $t \leq L$. A t-**path** c is a sequence of the form $c = (s_0, a_0, s_1, a_1 \ldots s_t)$ where s_i and a_i are states and actions in M. The t-**subpath** c_t of an L-path $(s_0, a_0, s_1, a_1 \ldots s_L)$ is $c_t = (s_0, a_0, s_1, a_1 \ldots s_t)$.

An algorithm is a mapping from the set of all paths with respect to M to the set of all actions. More formally,

DEFINITION 8.1.2. Let M be an L-epoch MDP. An **algorithm** $\mathcal{A}$ for M is a deterministic function:

$$\mathcal{A} : \{\text{paths}\} \to \{a_1, a_2, \ldots a_A\}$$

where $\{\text{paths}\}$ is the set of all $\{0, 1, \ldots T-1\}$-paths for M and $\{a_1, a_2, \ldots a_A\}$ is the set of all actions for M.

For simplicity, we only consider deterministic algorithms. Note that there is little difference between an algorithm and an "agent". However, in our terminology, an algorithm differs from a policy in that the algorithm has a "memory" where as a policy is "memoryless", *ie* the policy only maps a state-time to an action and disregards the previous experience.

An algorithm $\mathcal{A}$ along with an L-epoch MDP M and a starting state s_0 induces a distribution over L-paths. This distribution is analogous to the distribution over paths induced by a policy π (see section 2.1) and is defined as

$$\Pr(s_0, a_0, \ldots s_L | \mathcal{A}, M, s_0) \equiv \Pi_{\tau=0}^{L-1} I(\mathcal{A}(c_\tau) = a_\tau) P(s_{\tau+1} | s_\tau, a_\tau)$$

where $P(\cdot|s, a)$ is the transition model of M, and $I(\mathcal{A}(c_\tau) = a_\tau)$ is the indicator function which is 1 if $\mathcal{A}(c_\tau) = a_\tau$ and is 0 else. The indicator function is used since our algorithm is deterministic.

Under this distribution, given a t-path $(s_0, a_0, \ldots, s_t)$, the probability that $(s_t, a_t, \ldots s_L)$ is the remainder of the L-path is:

$$\Pr(s_t, a_t, \ldots s_L | \mathcal{A}, M, s_0, a_0, \ldots s_t) = \Pi_{\tau=t}^{L-1} I(\mathcal{A}(c_\tau) = a_\tau) P(s_{\tau+1} | s_\tau, a_\tau)\,.$$

Although the transition model obeys a Markov property, $\mathcal{A}$ does not necessarily obey such a property since this function could be dependent on the entire t-path $(s_0, a_0, \ldots, s_t)$. This distribution is useful when defining the value of an algorithm for a t-path.

8.2. Optimality Criteria

The situation we are considering is one in which $\mathcal{A}$ is run in M starting from s_0 and an L-path c is obtained, *ie* c is sampled from $\Pr(\cdot|M, \mathcal{A}, s_0)$. We desire a notion of optimality that reflects the algorithm $\mathcal{A}$'s behavior. The notion optimality considered is one with respect to an imposed planning horizon imposed by T of γ. Informally, we say that $\mathcal{A}$ is ε near-optimal at time t if the *value* of $\mathcal{A}$ at time t is ε close to the optimal value, with respect to T or γ. We start with the γ-discounted case since the notion is more straightforward than the T-step case.

8.2.1. γ-Discounted Optimality on a Path. Let us define the value of an algorithm on a t-path. Informally, it is the expected discounted reward of our algorithm from time t onward, assuming that the initial sequence of state-actions is the t-path $(s_0, a_0, \ldots s_t)$. The following definition uses the probability $\Pr(\cdot|\mathcal{A}, M, s_0, a_0, \ldots s_t)$ which was defined in the last section.

DEFINITION 8.2.1. Let M be an infinite horizon MDP. The **γ-discounted value** of algorithm $\mathcal{A}$ with respect to M from a t-path $(s_0, a_0, \ldots s_t)$ is:

$$U_{\mathcal{A},\gamma,M}(s_0, a_0, \ldots s_t) \equiv (1-\gamma) E_{(s_t, a_t, \ldots) \sim \Pr(\cdot|\mathcal{A}, M, s_0, a_0, \ldots s_t)} \left[\sum_{\tau=t}^{\infty} \gamma^{\tau - t} r(s_\tau, a_\tau) \right]$$

Let Π be the class of all algorithms for M. The **optimal γ-discounted value** from a t-path c is:

$$U^*_{\gamma,M}(c) \equiv \sup_{\mathcal{A} \in \Pi} U_{\mathcal{A},\gamma,M}(c) \,.$$

We drop the M dependence when clear from context. Due to the Markov property, we know that $U^*_\gamma(c) = V^*_\gamma(s)$, where s is the last state in the path. Since a policy π is a memoryless algorithm, then

$$U_{\pi,\gamma}(s_0, a_0, \ldots s_t) = V_{\pi,\gamma}(s_t)$$

which follows form the Markov property.

A notion of optimality is as follows. Let $c = (s_0, a_0, s_1, a_1 \ldots)$ be an ∞-path sampled from $\Pr(\cdot|\mathcal{A}, M, s_0)$, where M is an infinite horizon MDP.[2] Recall that a subpath c_t is just the subsequence $(s_0, a_0, \ldots s_t)$. We say $\mathcal{A}$ is ε near-optimal *at time* t if

$$U_{\mathcal{A},\gamma}(c_t) \geq U^*_\gamma(c_t) - \varepsilon \,.$$

Note the somewhat self-referential nature of this statement, since the algorithm produces the path c and optimality at time t is judged with respect to the value of the algorithm on the subpath c_t.

Clearly, it is not feasible for $\mathcal{A}$ to be near-optimal at all times t, if $\mathcal{A}$ has no knowledge of the MDP. We refer to the number of timesteps in which the previous condition is not satisfied as the *sample complexity of exploration*. Intuitively, it is the number of states at which the agent is not exploiting for near-optimal reward with respect to the horizon imposed by γ.

8.2.2. T-step Optimality on a Path. Defining a similar notion for the T-step case requires some additional technicalities. Unlike in the discounted case, a T-step optimal policy is non-stationary. Crudely, our notion of a good algorithm is one which executes a T-step, near-optimal policy for every cycle of T-steps. Consider a counter which counts as follows

$$0, 1, \ldots, T-1, 0, 1, \ldots T-1, 0, 1, 2, \ldots \,.$$

ie the counter value at time t is $t \mathrm{mod} T$. We say t' is the T-step end time of time t if t' is the next time at which the counter value is 0. Hence the time $2T$ is the T-step end time for all of the times $T, T+1, \ldots, 2T-1$. The value of $\mathcal{A}$ for a t-path is the (normalized) expected reward obtained from time t up until the T-step end time of t. More formally,

[2] See Puterman [1994] for a more formal definition of the probability of ∞-paths in infinite horizon setting.

DEFINITION 8.2.2. The time t' is the **T-step end time** of time t, if t' is the smallest time such that $t' \text{mod} T = 0$ and $t' > t$. Let M be an L-epoch MDP, and let t' be the T-step end time for time t. The **T-step undiscounted value** of an algorithm $\mathcal{A}$ from a path $(s_0, a_0, \ldots s_t)$ is:

$$U_{\mathcal{A},M}(s_0, a_0, \ldots s_t) \equiv \frac{1}{T} E_{(s_t, a_t, \ldots s_L) \sim \Pr(\cdot | \mathcal{A}, M, s_0, a_0, \ldots s_t)} \left[\sum_{\tau=t}^{t'-1} r(s_\tau, a_\tau) \right] .$$

Let Π be the class of all algorithms for M. The **optimal T-step undiscounted value** from a t-path c is:

$$U^*_M(c) \equiv \sup_{\mathcal{A} \in \Pi} U_{\mathcal{A},M}(c) .$$

Again, we normalize these values by T. Intuitively, $U_{\mathcal{A}}(s_0, a_0, \ldots s_t)$ is the sum (normalized) reward obtained until the T-step cycle is over. This is the analogous definition to $U_{\mathcal{A},\gamma}(s_0, a_0, \ldots s_t)$. Again, due to the Markov property, we know that $U^*(s_0, a_0, \ldots s_t)$ only depends on the last state s_t.

A notion of optimality similar to the γ-discounted case is as follows. Let $c = (s_0, a_0, \ldots s_L)$ be an L-path sampled according to $\Pr(\cdot | \mathcal{A}, M, s_0)$. We say $\mathcal{A}$ is *ε near-optimal at time t* if

$$U_{\mathcal{A}}(c_t) \geq U^*(c_t) - \varepsilon .$$

where c_t is the subpath $(s_0, a_0, \ldots s_t)$. Again, it is not feasible for $\mathcal{A}$ to be near-optimal at all times t if $\mathcal{A}$ has no knowledge of the MDP. As in the discounted case, we refer to the number of timesteps in which the previous condition is not satisfied as the *sample complexity of exploration.*

8.3. Main Theorems

This section presents the main claims of this chapter, which are upper and lower bounds on the sample complexity of exploration. We begin with the results on general MDPs and then state the results for deterministic MDPs. Our analysis focuses on the T-step case, and for the discounted case, we only state an upper bound.

8.3.1. General MDPs. The first theorem is an upper bound on the sample complexity of exploration for the T-step case. Recall each timestep corresponds to one sample transition.

THEOREM 8.3.1. *(T-step sample complexity) Let M be an L-epoch MDP and s_0 be a state for M. There exists an algorithm $\mathcal{A}$ taking inputs N, A, T, ε, and δ, such that if c is an L-path sampled from $\Pr(\cdot | \mathcal{A}, M, s_0)$, then with probability greater than $1 - \delta$, the statement*

$$U_{\mathcal{A}}(c_t) \geq U^*(c_t) - \varepsilon$$

is true for all but $O(\frac{N^2 A T^3}{\varepsilon^3} \log^2 \frac{NA}{\delta})$ timesteps $t \leq L$.

Crucially, note that this number of non-near optimal steps does not depend on L. Thus, regardless of how long the algorithm is run, there is only a fixed (polynomial) number of steps in which $\mathcal{A}$ is not acting near-optimally. This guarantee does not imply that the agent finds a near-optimal policy from every state, but only that it is executing near-optimal sequences of actions (with respect to T) from those states it happens to visit. Fixing other

variables and ignoring log factors, this bound is $O(N^2 A)$, which is precisely the number of parameters used to specify the transition model.

The following theorem is for the discounted case (where $L = \infty$). For comparison purpose, we state the result in terms of $T_\varepsilon = \frac{\log \varepsilon}{1-\gamma}$, which is the cutoff time in which the infinite-horizon, γ-discounted value function is $O(\varepsilon)$ close to a value function with cutoff time T_ε (see section 2.3.3)

THEOREM 8.3.2. *(γ-discounted sample complexity) Let M be an infinite horizon MDP and s_0 be a state for M. There exists an algorithm $\mathcal{A}$ taking inputs N, A, γ, ε, and δ, such that if c is an ∞-path sampled from $\Pr(\cdot|\mathcal{A}, M, s_0)$, then with probability greater than $1-\delta$, the statement*

$$U_{\mathcal{A},\gamma}(c_t) \geq U^*_\gamma(c_t) - \varepsilon$$

is true for all but $O(\frac{N^2 A T_\varepsilon^3}{\varepsilon^3} \log^2 \frac{NA}{\delta})$ timesteps, where $T_\varepsilon = \frac{\log \varepsilon}{1-\gamma}$.

This guarantee is parsimonious with the T-step undiscounted case, since in this case there is no L dependence (so we could set $L = \infty$ to obtain an analogous statement).

As mentioned earlier, these guarantees are different than the original E^3 guarantee which referred to mixing times and made ergodicity assumptions. For the T-case, the original statement is of the form that L needs to be polynomial in the appropriate quantities before the return reaped by the agent is close to the optimal return among those policies that mix in time T. Here, we allow L to be arbitrary and make our statements with respect to the expected value of the algorithm itself (rather than the return received). Note that if L is chosen to be sufficiently large, the time spent on "exploration" is a small fraction of the total time L. The statement referring to mixing times follows in a straightforward manner from this fact with appropriate additional ergodicity assumptions.

For γ-discounted case, the original guarantee is of the form that the E^3 algorithm only finds a single state at which it obtains a near-optimal policy from that state. This guarantee is stronger since the algorithm $\mathcal{A}$ is near-optimal from *all* states it visits *except* for $O(\frac{N^2 A T_\varepsilon^3}{\varepsilon^3} \log^2 \frac{NA}{\delta})$ states. One can easily use this theorem to make a guarantee that compares the return reaped by $\mathcal{A}$ to the return of those policies that mix in $O\left(\frac{1}{1-\gamma}\right)$ time (see see Baxter and Bartlett [2001] and Kakade [2001] for notions of "γ mixing").

In addition to this sample complexity statement, the following theorem bounds the *total* number of dynamic programming computations performed by $\mathcal{A}$ on an MDP of state-action size $N \times A$. Here, we say a "table lookup" is just accessing a table stored in memory of size N by A.

THEOREM 8.3.3. *(Computational Complexity) There exists an algorithm $\mathcal{A}$ that satisfies the conditions in theorem 8.3.1, and uses a total number of dynamic programming computations that is bounded by N and a total number of "table lookups" that is $O(L)$.*

Similar to the sample complexity result, the *total* number of dynamic programming computations performed by the algorithm does not depend on L. Only the number of "table lookups" depends on L which is due to executing a policy stored in memory for $O(L)$ steps. Note that the original algorithm does not guarantee that the amount of off-line computation is independent of the run time. The modification we make to the algorithm is intuitive. The algorithm simply caches the exploitation and exploration policies at states

where it has learned an accurate exploitation policy and uses a "table lookup" to execute the policy at these states.

These sample complexity results are intuitively appealing since N^2A is the number of parameters used to specify the transition matrix. Unfortunately, there is gap between this upper bound and the following lower bound.

THEOREM 8.3.4. *(Lower Bound) For any algorithm $\mathcal{A}$, there exists an L-epoch MDP M and a state s_0 such that if c is an L-path sampled from $\Pr(\cdot|\mathcal{A}, M, s_0)$, then with probability greater than $1-\delta$, the statement*

$$U_{\mathcal{A}}(c_t) \geq U^*(c_t) - \varepsilon$$

is false for $\Omega(\frac{NAT}{\varepsilon}\log\frac{1}{\delta})$ timesteps $t \leq L$.

Recall that lower bound is identical to the lower bound on the sample complexity required to compute an optimal policy (at a single state) with access to a generative model (see section 2.5). Importantly, note that this lower bound suggests that an accurate estimate of the transition model is *not* required to obtain a near-optimal policy. We return to the issue of accurate model building in the next chapter.

8.3.2. Deterministic MDPs. A deterministic MDP is one which has deterministic transition probabilities and rewards. For these MDPs, optimal T-step polices can be found exactly so we don't concern ourselves with probably approximately correct statements.

The results for deterministic MDPs are similar to those in Koenig and Simmons [1993], except we explicitly state the dependencies in terms of N, A, and T.

THEOREM 8.3.5. *(Deterministic Sample Complexity) Let M be an L-epoch deterministic MDP and s_0 be a state in M. There exists an algorithm $\mathcal{A}$ such that if c is an L-path sampled from $\Pr(\cdot|\mathcal{A}, M, s_0)$, then the statement*

$$U_{\mathcal{A}}(c_t) = U^*(c_t)$$

is true for all but NAT timesteps $t \leq L$.

Unlike in the stochastic case, the lower bound matches the upper bound.

THEOREM 8.3.6. *(Deterministic Lower Bound) For any algorithm $\mathcal{A}$, there exists an L-epoch MDP M and a state s_0 in M such that if c is an L-path sampled from $\Pr(\cdot|\mathcal{A}, M, s_0)$, then the statement*

$$U_{\mathcal{A}}(c_t) = U^*(c_t)$$

is false for $\Omega(NAT)$ timesteps $t \leq L$.

8.4. The Modified R_{max} Algorithm

This section specifies the R_{max} algorithm, which is a generalization of the E^3 algorithm. Both algorithms are model based algorithms, *ie* the algorithms estimate a transition model using the experience obtained through acting in the MDP. Both algorithms then use this empirical model for both exploration and exploitation purposes. The key insight to both algorithms is that exploration can be done efficiently when the agent is not exploiting. The R_{max} algorithm handles the tradeoff between exploration and exploitation in a more natural way than its precursor, and this additional modification provides the leverage in making our more general performance guarantees.

A crucial notion in both algorithms is that of a *known* state — a state visited often enough such that the estimated transition model for that state is "close" to its true values in M.

DEFINITION 8.4.1. A state is m-**known** if each of its actions has been tried m times.

We typically just say known rather than m-known. It turns out that in this analysis the value of m is the gap between our upper and lower bounds. Later we take care in specifying what constitutes "close" and in what value of m is sufficient to obtain an accurate model with high probability.

The algorithm makes a distinction between the known and unknown states. We refer to the currently known set by K. Successful exploration is when the agent visits a state not in K. When an unknown state is visited, the agent engages in **balanced wandering** — it chooses the action at that state that it has tried the least often (and ties are broken randomly). Therefore, after mA visits to an unknown state, it becomes known, since each action has been tried m times. By the pigeon hole principle, successful exploration can only occur mNA times before all state-actions become known (though this does not necessarily occur).

The algorithm proceeds by using observations at the known states to construct an approximate MDP $\hat{M}_K$ at these states. To do this, the algorithm maintains the obvious statistics from its observed experience, and it constructs the approximate transition model with observed empirical frequencies. Estimating the reward function is trivial due to our assumption that the reward function is deterministic.

This approximate model is used for planning exploration and exploitation policies. In the R_{max} algorithm, $\hat{M}_K$ is altered such that all states not in K are absorbing and maximally rewarding (for $r = 1$ in our setting) — this construction can though of as *optimism in the face of uncertainty*. It turns out that an optimal policy $\hat{\pi}$ for $\hat{M}_K$ is either a "good" exploration or a "good" exploitation policy in M. A good exploitation policy in M is one which has near-optimal T-step reward and a good exploration policy in M is one which escapes from the set of known states K quickly.

This variant and analysis differ from E^3 and R_{max} in the following ways:

(1) a more general optimality guarantee is made
  (a) no notions of "mixing" are assumed (for the T case)
  (b) the discounted case has an analogous optimality guarantee
(2) computations are re-used.
(3) a tight bound on m is used, which requires
  (a) a less stringent l_1 accuracy condition on $\hat{P}_K$
  (b) a tight sample complexity result to satisfy this condition
(4) a more general "induced inequality" lemma is used for explore/exploit tradeoff
(5) a weaker accuracy condition (8.5.1) is explicitly stated for the algorithm

First, some definitions related to the value of a policy are useful. Then we are ready to present the key "induced inequality" lemma and the algorithm.

8.4.1. T-Step Values. Recall that the value $V_{\pi,t,M}(s)$ is the undiscounted return, of a policy π starting from state s, over the timeperiod starting at t and ending at L (since M is an L-epoch MDP). Here, π is a sequence of L decision rules.

The R_{max} algorithm uses deterministic T**-step policies** π, which are T-step sequences of decision rules of the form $(\pi(\cdot,0),\pi(\cdot,1),\ldots\pi(\cdot,T-1))$.[3] It is useful to define the t-value of a T-step policy in the L-epoch MDP M. The t-value is just the reward obtained from from time t up until time T from starting at state s and following $(\pi(\cdot,t),\pi(\cdot,t+1),\ldots\pi(\cdot,T-1))$. More formally,

DEFINITION 8.4.2. Let M be an L-epoch MDP and let π be a T-step policy for M. For a time $t<T$, the t**-value** $U_{\pi,t,M}(s)$ of π at state s is

$$U_{\pi,t,M}(s) \equiv \frac{1}{T} E_{(s_t,a_t,\ldots s_{T-1},a_{T-1})\sim \Pr(\cdot|\pi,M,s_t=s)} \left[\sum_{\tau=t}^{T-1} r(s_\tau,a_\tau)\right] .$$

Let Π be the class of all T-step policies for M. The **optimal t-value** of π at state s is:

$$U^*_{t,M}(s) \equiv \sup_{\pi\in\Pi} U_{\pi,t,M}(s) .$$

A T**-step optimal policy** π at state s is one such that

$$U_{\pi,t,M}(s) = U^*_{t,M}(s) .$$

As opposed to $V_{\pi,t,M}$, $U_{\pi,t,M}(s)$ is just the (normalized) sum reward from time t up until time T (rather than L) obtained under the execution of the T-step policy π for $T-t$ steps.

8.4.2. Explore or Exploit? The following definition of an induced MDP M_K with respect to M and a set of states K is useful. We define M_K and M to have the same state-action space. For all states in K, M_K is identical to M. For all states not in K, M_K modifies these states to be absorbing and maximally rewarding (for $r=1$).[4] More formally,

DEFINITION 8.4.3. Let M be an MDP with transition model P and let K be a set of states. The **induced MDP** M_K, with respect to K and M is defined as follows. M_K has the same number of epochs and the same state-action space as M. M_K has a transition model P_K and reward function r_K specified as follows.

If $s\in K$, for all actions a and states s'

$$\begin{aligned} P_K(s'|s,a) &\equiv P(s'|s,a) \\ r_K(s,a) &\equiv r(s,a) . \end{aligned}$$

If $s\notin K$, for all actions a

$$\begin{aligned} P_K(s|s,a) &\equiv 1 \\ r_K(s,a) &\equiv 1 . \end{aligned}$$

The following lemma on inequalities based on value functions for M and M_K provides the foundation for the algorithm. Here, $\Pr(\text{escape from } K|\pi,M,s_t=s)$ is the probability of reaching a state not in K in the path $(s_t,a_t\ldots,s_{T-1},a_{T-1})$ obtained while following π in M starting from state $s_t=s$, *ie*

$\Pr(\text{escape from } K|\pi,M,s_t=s) \equiv E_{(s_t,a_t\ldots,s_{T-1},a_{T-1})\sim\Pr(\cdot|\pi,M,s_t=s)}\left[I\left(\exists\tau \text{ s.t. } s_\tau\notin K\right)\right]$

where I is the indicator function denoting if some state s_τ is not in K.

[3]Technically, a policy is a rule for acting over the *entire* L-epoch MDP (see section 6.2), and this T-step policy is not a policy but a sequence of T decision rules.

[4]For simplicity, we avoid using a single additional absorbing state as in E^3 and R_{max}. Instead, we just make all states not in K absorbing. Hence, M and M_K have the same state space.

LEMMA 8.4.4. *(Induced Inequalities) Let M be an MDP, K be a set of states, and M_K be an induced MDP with respect to K and M. For all T-step policies π, times $t < T$, and states s,*

$$U_{\pi,t,M_K}(s) \geq U_{\pi,tM}(s)$$

and

$$U_{\pi,t,M}(s) \geq U_{\pi,t,M_K}(s) - \Pr(\text{escape from } K|\pi, M, s_t = s)$$

The first inequality trivially follows from the construction of M_K. The second inequality states that a policy in M has value close to that in M_K if the policy does not escape quickly (in T-steps) from K in M. An important point is that this guarantee is made simultaneously for *all* $t < T$. This is relevant to our notion of optimality since we desire the expectation of our algorithm to be near-optimal at *every* state encountered with respect to our imposed horizon time of T.

PROOF. The first inequality immediately follows since M_K and M are identical on K, and outside of K, M_K is absorbing and maximally rewarding.

To prove the second inequality, let $P_{\tau,M_K}(s)$ and $P_{\tau,M}(s)$ be the probability that the state at time τ is s while following π starting from s_t at time t in M_K and M, respectively. We slightly abuse notation and write $r_K(s,\tau) = r_K(s,\pi(s,\tau))$ and $r(s,\tau) = r(s,\pi(s,\tau))$ for the reward obtained at state-time (s,τ) by following the policy.

Following from the definition of M_K, for all $s \in K$, $P_{\tau,M_K}(s) \leq P_{\tau,M}(s)$. Also, for $s \in K$, $r_K(s,\tau) = r(s,\tau)$ and for $s \notin K$, $r_K(s,\tau) = 1$. Hence, for all $t \leq \tau < T$

$$\begin{aligned}
& E_{s\sim P_{\tau,M_K}}[r_K(s,\tau)] - E_{s\sim P_{\tau,M}}[r(s,\tau)] \\
= & \sum_{s\in K} P_{\tau,M_K}(s) r_K(s,\tau) - P_{\tau,M}(s) r(s,\tau) \\
& + \sum_{s\notin K} P_{\tau,M_K}(s) r_K(s,\tau) - P_{\tau,M}(s) r(s,\tau) \\
\leq & \sum_{s\in K} P_{\tau,M_K}(s)(r_K(s,\tau) - r(s,\tau)) + \sum_{s\notin K} P_{\tau,M_K}(s) r_K(s,\tau) \\
= & \sum_{s\notin K} P_{\tau,M_K}(s) \\
\leq & P(\text{escape from } K|\pi, M_K, s_t) \\
\leq & P(\text{escape from } K|\pi, M, s_t)\,.
\end{aligned}$$

Since

$$V_{\pi,t,M_K}(s_0) - V_{\pi,t,M}(s_0) = \frac{1}{T}\sum_{\tau=t}^{T-1}\left(E_{s\sim P_{\tau,M_K}}[r_K(s,\tau)] - E_{s\sim P_{\tau,M}}[r(s,\tau)]\right)$$

the second inequality now follows. □

To point out connections with both the E^3 and R_{max} explore or exploit lemma, we state the following corollary, which is a slightly stronger version, with respect to time, of the key lemma in R_{max} (Brafman and Tennenholtz [2001]). However, it is the previous lemma that is used in the analysis of our algorithm.

The corollary states that if we derive an optimal policy π from M_K then either π escapes quickly from K in M or else π is a near-optimal policy in M (with respect to T). Hence, by following the *single* policy π no explicit decision to explore or exploit needs to be made, since π is implicitly making this tradeoff.[5]

COROLLARY 8.4.5. *(Implicit Explore or Exploit) Let M be an MDP, K be a set of states, and M_K be an induced MDP with respect to K and M. Let π be an optimal T-step policy in M_K. For all states s and times $t \leq T$*

$$U_{\pi,t,M}(s) \geq U^*_{t,M}(s) - \Pr(\text{escape from } K|\pi, M, s_t = s)$$

PROOF. Let π^* be an optimal T-step policy in M. Using the fact that π is T-step optimal in M_K and using both inequalities in the previous lemma, we have for all $t < T$,

$$\begin{aligned} U_{\pi,t,M}(s) &\geq U_{\pi,t,M_K}(s_0) - \Pr(\text{escape from } K|\pi, M, s_t = s) \\ &\geq U_{\pi^*,t,M_K}(s_0) - \Pr(\text{escape from } K|\pi, M, s_t = s) \\ &\geq U^*_{t,M}(s_0) - \Pr(\text{escape from } K|\pi, M, s_t = s) \end{aligned}$$

where last step follows since $U_{\pi^*,t,M_K}(s_0) \geq U_{\pi^*,t,M}(s_0)$. □

8.4.3. The algorithm. Let K be an m-known set. If we knew the MDP M_K, then the last lemma suggests that this MDP is sensible to use for planning. Since the transition model of M_K is not known, we use an approximation to this MDP $\hat{M}_K$, with transition model $\hat{P}_K$. For states $s \in K$, define $\hat{P}_K$ follows,

$$\hat{P}_K(s'|s,a) = \frac{(\#\text{ of times } s \to s' \text{ under action a})}{\#\text{ of visits to } s}$$

and for states not in K, $\hat{P}_K$ is absorbing. Clearly, the value that is chosen for m determines the quality of this approximation. However, let us ignore this for now and present the algorithm in terms of the parameter m.

Algorithm 14 R_{max}

(1) Set $K = \emptyset$
(2) **Act**: Is $s \in K$?
  (a) Yes, execute the action $\hat{\pi}(s, t \bmod T)$. Goto 2.
  (b) No, perform balanced wandering for one timestep.
    (i) If a state becomes m-known, goto 3.
    (ii) Else, goto 2.
(3) **Compute**:
  (a) Update K and $\hat{M}_K$
  (b) Compute an optimal policy $\hat{\pi}$ for $\hat{M}_K$. Goto 2.

[5]The use of a *single* policy by R_{max} also allows us to make our stronger guarantees. The reason is because the guarantees of interest are with respect to the expected return of the algorithm itself. The use of a single policy means that the value of the algorithm is directly related to the value of the policy. In contrast, the E^3 algorithm switches between an explore and an exploit policy, so the expected value of the algorithm itself is not directly connected to the value of the policy, unless E^3 executes a *single* policy for T steps. However, executing a single policy for T steps makes it more tricky to make our guarantee that the algorithm is optimal from *every* subpath c_t. The author conjectures that the straightforward E^3 variant has a larger sample complexity bound in terms of T and ε.

The slightly modified R_{max} algorithm is shown in 14, where s is the current state and t is the current time (so $t \text{mod} T$ is the current cycle time). Initially K is empty, and so the algorithm first engages in balanced wandering, where the agent tries the action which has been taken the least number of times at the current state. This is continued until a state becomes m-known, which must occur by the pigeonhole principle. Upon obtaining a state that is m-known, the algorithm then updates K and $\hat{M}_K$ and an optimal policy $\hat{\pi}$ is computed for $\hat{M}_K$.

If the algorithm is at a state $s \in K$, then $\hat{\pi}$ is executed with respect to the T-step cycle (recall section 8.2.2), *ie* the action $\hat{\pi}(s, t \text{mod} T)$ is taken. At all times, if the agent ever reaches a state $s \notin K$, balanced wandering is resumed. If the known set K ever changes, then MDP $\hat{M}_K$ is updated and the policy $\hat{\pi}$ is recomputed. Note that computations are performed only when the known set K changes, else the algorithm just performs "table lookups".

We hope that $\hat{M}_K$ is a good approximation to M_K and that this implies the agent is either exploiting or efficiently exploring. By the Pigeonhole Principle, successful exploration can only occur mNA times. Hence, as long as the escape probability is "large", exploration must "quickly" cease and exploitation must occur (as suggested by the explore or exploit corollary, 8.4.5).

8.5. The Analysis

This section provides the analysis of the upper bounds. In the first subsection, we assume that a value of m is chosen large enough to obtain sufficiently accurate $\hat{M}_K$'s and a sample complexity bound is stated in terms of m. It turns out that m is essentially the gap between the lower and upper bounds, so we desire a tight analysis to determine an appropriate value of m. The following subsection provides an improved l_1 accuracy condition for determining if $\hat{M}_K$ is accurate. It turns out that a straightforward Chernoff bound analysis is not sufficient to obtain a tight bound and a more involved analysis is used to determine m. The final subsection completes the proofs of the upper bounds for both general and deterministic MDPs.

An important point to note is that the accuracy condition we state in the first subsection is, informally, the weakest condition needed for our analysis to go through. This accuracy condition does *not* necessarily imply that we need to obtain an accurate transition model for $\hat{M}_K$, only that optimal policies derived from $\hat{M}_K$ are accurate. However, when determining m in the subsection thereafter, we actually ensure that the transition model for $\hat{M}_K$ is an accurate estimate to that in M_K (though this is done in a weaker l_1 sense as compared to the original E^3 and R_{max} algorithms). This issue of "accurate model building" lies at the heart of the gap between our lower and upper bound and is discussed in the next chapter, where we examine "model based" based approaches more carefully.

8.5.1. The Sample Complexity in terms of m. For now let us just assume a value of m is chosen such that $\hat{M}_K$ is accurate in the following sense.

CONDITION 8.5.1. (Approximation Condition) If R_{max} uses the set of states K and an MDP $\hat{M}_K$, then for the optimal policy $\hat{\pi}$ for $\hat{M}_K$ assume that for all states s and times $t < T$

$$U_{\hat{\pi},t,M_K}(s) \geq U^*_{t,M_K}(s) - \varepsilon$$

The assumption states that the policy $\hat{\pi}$ that our algorithm derives from $\hat{M}_K$ is near-optimal in M_K. Informally, this is the weakest condition needed in order for the analysis to go through. Note that this condition does not state we require an accurate transition model of M_K. In the next subsection, we determine a value of m such that this condition holds with high probability.

The following lemma bounds the sample complexity of exploration in terms of m.

LEMMA 8.5.2. *Let M be an L-epoch MDP and s_0 be a state for M. If c is an L-path sampled from $\Pr(\cdot|R_{max}, M, s_0)$ and if condition 8.5.1 holds, then with probability greater than $1-\delta$, the statement*

$$U_{R_{max}}(c_t) \geq U^*(c_t) - 2\varepsilon$$

is true for all but $O(\frac{mNAT}{\varepsilon} \log \frac{NA}{\delta})$ timesteps $t \leq L$.

The high-level idea of the proof is as follows. Under the induced inequality lemma (8.4.4) and the previous condition, then we can show that either $\hat{\pi}$ escapes K in M with probability greater than an ε or $\hat{\pi}$ is a 2ε near-optimal policy in M (where one factor of ε is due to the accuracy assumption and the other factor is due to having an escape probability less than ε). By the Pigeonhole Principle, successful exploration can only occur mNA times, so eventually exploitation must occur.

PROOF. Let $\hat{\pi}$ be an optimal policy with respect to some $\hat{M}_K$ that is used by R_{max}. Let π^* be an optimal policy in M. The induced inequality lemma (8.4.5) and condition 8.5.1 imply that for all $t \leq T$ and states s

$$\begin{aligned} U_{\hat{\pi},t,M}(s) &\geq U_{\hat{\pi},t,M_K}(s) - \Pr(\text{escape from } K|\hat{\pi}, M, s_t = s) \\ &\geq U^*_{t,M_K}(s) - \varepsilon - \Pr(\text{escape from } K|\hat{\pi}, M, s_t = s) \\ &\geq U_{\pi^*,t,M_K}(s) - \varepsilon - \Pr(\text{escape from } K|\hat{\pi}, M, s_t = s) \\ &\geq U^*_{t,M}(s) - \varepsilon - \Pr(\text{escape from } K|\hat{\pi}, M, s_t = s) \end{aligned}$$

where we have used both inequalities in the induced inequality lemma and have used the optimality of π^* in M.

Recall that R_{max} executes the policy $\hat{\pi}$ in sync with the T-step cycle time as along as $s \in K$. The definition of $U_{\mathcal{A}}$ and the previous inequality imply that either

$$U_{R_{max}}(c_t) \geq U^*(c_t) - 2\varepsilon$$

or the probability that $\hat{\pi}$ escapes from K before the T-end time for t must be greater than ε.

Now, we address how many timesteps the T-step escape probability can be greater than ε for any K. Each attempted exploration can be viewed as a Bernoulli trial with chance of success greater than ε. There can be at most mNA successful exploration attempts, until all state-actions are known and the escape probability is 0. Note that in $\frac{mNA}{\varepsilon}$ steps the mean number of successful exploration attempts is mNA. The Hoeffding's bound states that we need $\frac{1}{\beta^2} \log \frac{1}{\delta}$ samples to obtain a β fractionally accurate estimate of the mean, where each "sample" is $\frac{mNA}{\varepsilon}$ attempts. Hence, we can choose a β such that $O(\frac{mNA}{\varepsilon} \log \frac{1}{\delta})$ attempts are sufficient for all mNA exploration attempts to succeed, with probability greater than $1-\delta$. Since each attempted exploration takes at most T steps, the number of exploration steps is bounded by $O(\frac{mNAT}{\varepsilon}) \log \frac{1}{\delta}$. □

Note that our lower bound is $\Omega(\frac{NAT}{\varepsilon}\log\frac{1}{\delta})$, and so the gap between our lower and upper bound is essentially m. The issue we now address is what is the sample size m. Clearly, a tight bound is desired here to minimize this gap.

8.5.2. What is the appropriate value of m? First, an approximation condition for $\hat{M}$ is provided, such that the approximation condition 8.5.1 holds. We use the following definition for our less stringent l_1 accuracy condition.

DEFINITION 8.5.3. (l_1 accuracy) We say that a transition model $\hat{P}$ is an ε-**approximation** to a transition model P if for all states s and actions a

$$\sum_{s'} |\hat{P}(s'|s,a) - P(s'|s,a)| < \varepsilon\,.$$

The following lemma addresses our accuracy needs.

LEMMA 8.5.4. *(ε-Approximation Condition) Let M and $\hat{M}$ be two MDPs with the same reward functions and the same state-action space. If the transition model of $\hat{M}$ is an ε-approximation to that of M, then for all T-step policies π, states s, and times $t < T$,*

$$|U_{\pi,t,\hat{M}}(s) - U_{\pi,t,M}(s)| < \varepsilon T\,.$$

As mentioned earlier, this guarantee is actually *stronger* than what is needed for our accuracy condition, since it holds for *all* policies. We address this issue in the next chapter.

PROOF. Let $S_{T-1} = (s, s_{t+1}, \ldots, s_{T-1})$ be a $T-t$ length sequence of states starting with a fixed state s and let S_τ be the subsequence $S_\tau = (s, s_{t+1}, \ldots, s_\tau)$. Let $\Pr(S_\tau)$ and $\widehat{\Pr}(S_\tau)$ be the probability of S_τ in M and $\hat{M}$, respectively, under policy π starting from $s_t = s$ at time t. Let $R(S_{T-1}) = \frac{1}{T}\sum_{\tau=t}^{T-1} r(s_\tau, \pi(s_\tau))$ and so

$$\begin{aligned} |U_{\pi,t,\hat{M}}(s) - U_{\pi,t,M}(s)| &= |\sum_{S_{T-1}} (\Pr(S_{T-1}) - \widehat{\Pr}(S_{T-1}))R_t(S_{T-1})| \\ &\leq \sum_{S_{T-1}} |\Pr(S_{T-1}) - \widehat{\Pr}(S_{T-1})| \end{aligned}$$

since R is bounded between 0 and 1.

We now show that the error between $\Pr(S_{T-1})$ and $\widehat{\Pr}(S_{T-1})$ is bounded as follows

$$\sum_{S_{T-1}} |\Pr(S_{T-1}) - \widehat{\Pr}(S_{T-1})| \leq \varepsilon T$$

where the sum is over all $T-t$ length sequences S_{T-1} that start with state $s_t = s$. Let P and $\hat{P}$ be the transition models in M and $\hat{M}$ respectively. Slightly abusing notation, let $P(s'|S_\tau) = P(s'|s_\tau, \pi(s_\tau))$ and $\hat{P}(s'|S_\tau) = \hat{P}(s'|s_\tau, \pi(s_\tau))$. The ε-approximation

condition implies that $\sum_{s'} |\hat{P}(s'|S_\tau) - P(s'|S_\tau)| \leq \varepsilon$. For any $\tau < \mathrm{T} - 1$, it follows that

$$\begin{aligned}
& \sum_{S_{\tau+1}} |\Pr(S_{\tau+1}) - \widehat{\Pr}(S_{\tau+1})| \\
= & \sum_{S_\tau, s'} |\Pr(S_\tau) P(s'|S_\tau) - \widehat{\Pr}(S_\tau) \hat{P}(s'|S_\tau)| \\
\leq & \sum_{S_\tau, s'} |\Pr(S_\tau) P(s'|S_\tau) - \widehat{\Pr}(S_\tau) P(s'|S_\tau)| + |\widehat{\Pr}(S_\tau) P(s'|S_\tau) - \widehat{\Pr}(S_\tau) \hat{P}(s'|S_\tau)| \\
= & \sum_{S_\tau} |\Pr(S_\tau) - \widehat{\Pr}(S_\tau)| \sum_{s'} P(s'|S_\tau) + \sum_{S_\tau} \widehat{\Pr}(S_\tau) \sum_{s'} |P(s'|S_\tau) - \hat{P}(s'|S_\tau)| \\
\leq & \sum_{S_\tau} |\Pr(S_\tau) - \widehat{\Pr}(S_\tau)| + \varepsilon \,.
\end{aligned}$$

Recursing on this equation leads to the result. □

The following technical lemma addresses how large m needs to be such that $\hat{P}$ is an ε-approximation to P. The following lemma is for an arbitrary distribution p over N elements.

LEMMA 8.5.5. *Assume that m samples are obtained from a distribution p, which is over a set of N elements. Let $\hat{p}$ be the empirical distribution, ie $\hat{p}(i) = \frac{\#\text{ of } i\text{'s observed}}{m}$. If $m = O(\frac{N}{\varepsilon^2} \log \frac{N}{\delta})$, then with probability greater than $1 - \delta$*

$$\sum_i |\hat{p}(i) - p(i)| \leq \varepsilon \,.$$

A straightforward Chernoff bound analysis is not sufficient to prove this result (as this would lead to an $O(N^2)$ result). The proof demands different fractional accuracies for different values of $p(i)$.

PROOF. For simplicity, we write p_i for $p(i)$ and $\hat{p}_i$ for $\hat{p}(i)$. The form of the Chernoff bound that we use is

$$P(|\hat{p}_i - p_i| > \alpha p_i) \leq 2 \exp(-\alpha^2 p_i m / 2) \,.$$

Let us define the values α_i as follows, based on the values p_i as follows:

$$\alpha_i = \begin{cases} \frac{\varepsilon}{2} & \text{if } p_i \geq \frac{1}{N} \\ \frac{\varepsilon}{2 p_i N} & \text{else} \end{cases}$$

Let us assume that for all i that

$$|\hat{p}_i - p_i| \leq \alpha_i p_i \,. \tag{8.5.1}$$

This implies that

$$\begin{aligned}\sum_i |\hat{p}(i) - p(i)| &\leq \sum_i \alpha_i p_i \\ &\leq \sum_{i \text{ st } p_i \geq \frac{1}{N}} \frac{\varepsilon}{2} p_i + \sum_{i \text{ st } p_i < \frac{1}{N}} \frac{\varepsilon}{2p_i N} p_i \\ &= \frac{\varepsilon}{2} \sum_{i \text{ st } p_i \geq \frac{1}{N}} p_i + \frac{\varepsilon}{2} \sum_{i \text{ st } p_i < \frac{1}{N}} \frac{1}{N} \\ &\leq \frac{\varepsilon}{2} + \frac{\varepsilon}{2}\end{aligned}$$

and so it is sufficient to show equation 8.5.1 holds with probability greater than $1 - \delta$ for $m = O(\frac{N}{\varepsilon^2} \log \frac{N}{\delta})$.

By the union bound and the Chernoff bound,

$$\begin{aligned}P(\exists i \text{ s.t. } |\hat{p}_i - p_i| > \alpha_i p_i) &\leq 2 \sum_i \exp(-\alpha_i^2 p_i m/2) \\ &\leq 2 \sum_{i:\ p_i \geq \frac{1}{N}} \exp(-\alpha_i^2 p_i m/2) + 2 \sum_{i:\ p_i < \frac{1}{N}} \exp(-\alpha_i^2 p_i m/2)\,.\end{aligned}$$

Using the value of α_i stated above, it follows that for i such that $p_i \geq \frac{1}{N}$, $\exp(-\alpha_i^2 p_i m/2) \leq \exp(-\frac{\varepsilon^2 p_i m}{8}) \leq \exp(-\frac{\varepsilon^2 m}{8N})$. Similarly, for i such that $p_i < \frac{1}{N}$, $\exp(-\alpha_i^2 p_i m/2) \leq \exp(-\frac{\varepsilon^2 m}{8N})$. Therefore,

$$\begin{aligned}P(\exists i \text{ st } |\hat{p}_i - p_i| > \alpha_i p_i) &\leq 2 \sum_i \exp(-\frac{\varepsilon^2 m}{8N}) \\ &= 2N \exp(-\frac{\varepsilon^2 m}{8N})\end{aligned}$$

and the result follows if we demand $2N \exp(-\frac{\varepsilon^2 m}{8N}) \leq \delta$. □

The previous two lemmas directly imply the following tightened result for m over the E^3 and R_{max} algorithm.

LEMMA 8.5.6. *(Setting m) In order for the approximation condition to hold (equation 8.5.1) with probability of error less than δ, it is sufficient that $m = O(\frac{NT^2}{\varepsilon^2} \log \frac{NA}{\delta})$.*

PROOF. By lemma 8.5.4, we need to set the l_1 error of $\hat{P}(|s,a)$ to be less than $\frac{\varepsilon}{2T}$ for all s and a. This implies that an optimal policy $\hat{\pi}$ in $\hat{M}_K$ has value in M_K that is ε close to the optimal value in M_K. There are NA of these transition probabilities, so if we allocate $\frac{\delta}{NA}$ error probability to each one, then the total error probability is less than δ. The result follows from the the previous lemma with $\varepsilon \leftarrow \frac{\varepsilon}{2T}$ and $\delta \leftarrow \frac{\delta}{NA}$. □

8.5.3. Putting the lemmas together. The proof of the main sample complexity upper bound for the T-case follows.

PROOF. (of theorem 8.3.1) For the T-step case, lemma 8.5.2 and lemma 8.5.6 directly imply the theorem with the alteration $\varepsilon \leftarrow \frac{\varepsilon}{4}$ and $\delta \leftarrow \frac{\delta}{2}$. □

The proof for the discounted case requires a few alterations, which we now outline.

PROOF. (of theorem 8.3.2) The γ-discounted case requires a few straightforward alterations to the lemmas. Note that if we choose $T_\varepsilon = O(\frac{\log \varepsilon}{1-\gamma})$ then the discounted return in time T_ε will be ε close to the infinite horizon discounted return. The induced inequality lemma can be modified to show that for any policy π

$$U_{\pi,\gamma,M}(s) \geq U_{\pi,\gamma,M_K}(s) - \Pr(\text{escape from } K \text{ in time } T_\varepsilon | \pi, M, s_t = s) - O(\varepsilon) .$$

We must also modify the R_{max} algorithm (14) as follows. Instead of computing a non-stationary policy $\hat{\pi}$, compute a stationary, γ-discounted optimal policy $\hat{\pi}$ for $\hat{M}_K$. Then at each time R_{max} executes $\hat{\pi}$ at state s, it chooses the action $\hat{\pi}(s)$ (which does not depend on any cycle time). Using these modifications, the proof parallels the T-case. □

The proof that the number of dynamic programming steps is $2N$ is straightforward.

PROOF. (of theorem 8.3.3) The Update line in the R_{max} algorithm (14) is only called when K changes. K only monotonically increases for N steps, which proves the first claim. The remainder of the steps only use "table lookups". □

For deterministic MDPs, we can use the same R_{max} algorithm by setting $m = 1$. The proof of the upper bound for deterministic MDPs is straightforward.

PROOF. (of theorem 8.3.5) It is clear that if $m = 1$ then $\hat{M}_K$ is a perfect approximation to M_K. Since the MDP is deterministic, any attempted escape from K in t steps succeeds with probability 1. By the pigeon hole principle, there are at most NA attempted explorations. Since each exploration is successful and could take at most T steps, there are at most NAT steps spent exploring. All other steps must be spent executing T-step optimal policies. □

8.6. Lower Bounds

Now we prove the lower bound for the stochastic case. The proof is an extension of the proof given for the lower bound on the number of calls to the generative model required to find a near-optimal policy, which was provided in section 2.5. Recall that this lower bound (theorem 2.5.2) was $\Omega(\frac{NAT}{\varepsilon} \log \delta)$.

Let us review the simple two state MDP used in the proof (shown in figure 8.6.1). State 1 is the only maximally rewarding (absorbing) state. In state 2, all but one action is absorbing. For this one action, the probability of a transition to the rewarding state is ε/T, else a self-transition occurs. The optimal policy has a reward of $\Omega(\varepsilon)$ from state 2, since the probability of transitioning to the maximally rewarding state is $\Omega(\varepsilon)$. To discover the optimal action at state 2 requires $\Omega(\frac{AT}{\varepsilon} \log \frac{1}{\delta})$ actions, and while this action is not discovered the agent cannot act near-optimally (see subsection 2.5.3).

PROOF. (of theorem 8.3.4) Extend the previous 2-state MDP to an N-state MDP as follows (see figure 8.6.1B). At any state $i > 1$, $A - 1$ of the actions are self transitions and the remaining action has a probability $\frac{\varepsilon}{T}$ of entering state 1 (else a self transition occurs). All states $i > 1$ are not rewarding. State 1 is maximally rewarding. If state 1 were absorbing, then to act optimally for any particular state $i > 1$, the agent must discover the rewarding action at this state which requires observing $\Omega(\frac{AT}{\varepsilon} \log \delta)$ calls. However, as it stands, once the agent enters state 1, then, trivially, an optimal policy is executed since state 1 is absorbing.

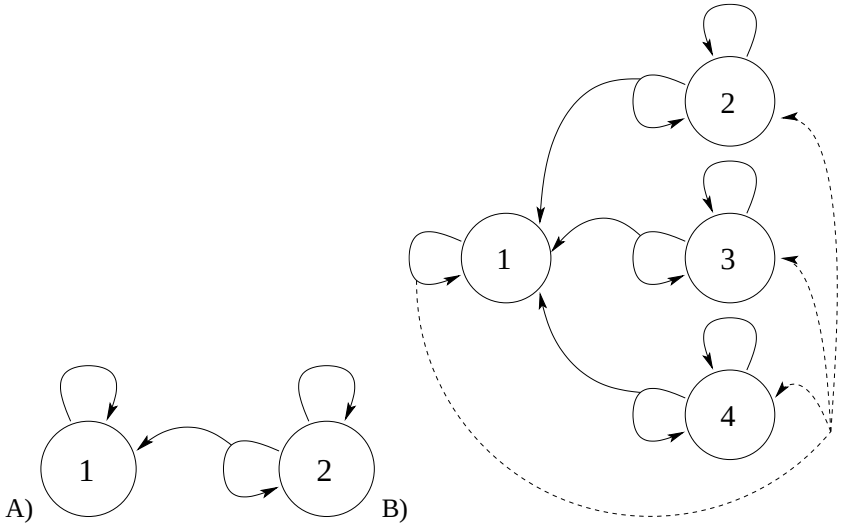

FIGURE 8.6.1. Stochastic MDPs. See text for description.

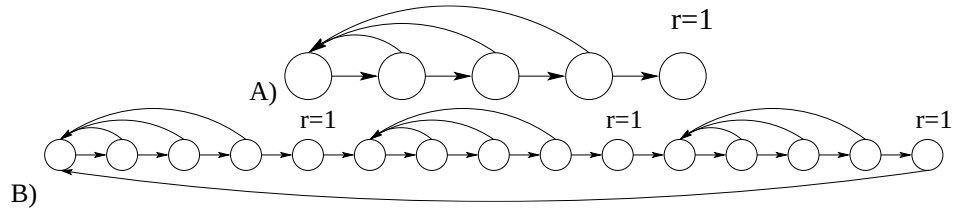

FIGURE 8.6.2. Deterministic MDPs A) Here, $N = 5$, $T = 5$, and $A = 2$ and the rightmost state is rewarding. B) Here, $N = 15$ and three copies of the MDP in A) are joined as shown. If $\frac{N}{T}$ is not an integer, then additional dummy states can be added in the construction, where each dummy state moves to the next dummy state (and then back to leftmost state).

Modify state 1 such that with probability $1/T^2$, the agent transitions uniformly to any other state. This does not alter the optimal value of $\Omega(\varepsilon)$ at states $i \geq 1$. However, for a sufficiently large L, the agent will eventually reach all states. Hence, $\Omega(\frac{NAT}{\varepsilon} \log \delta)$ transitions must be spent on discovering the optimal actions, while not executing a near-optimal policy. □

We now prove the lower bound for the deterministic case (which is related to the lower bound in Koenig and Simmons [1993]). Again, we exploit L being arbitrary.

PROOF. (of theorem 8.3.6) First, consider the simple case where $N = T$ in the figure shown in 8.6.2A. Here, only one action takes the agent to the right and all other actions return the agent to the leftmost state. The only reward is at the rightmost state. The actions in the MDP shown in the figure are unlabeled. Using the algorithm, we can label the actions such that the algorithm must try every action at every state, requiring NA steps, else the algorithm will fail to discover the rewarding state. Each action returns the agent

back to the left most state, and so $\Omega(NAT) = \Omega(T^2A)$ samples are required for A to find a near-optimal policy.

For the general case consider joining $\lfloor \frac{N}{T} \rfloor$ of these T-state MDPs, as shown in figure 8.6.2B (see figure caption for description). If $\frac{N}{T}$ is not an integer, dummy states can be used as shown in the figure. Here, each rewarding state takes the agent to the next set of T-states. If L is sufficiently large, then A must find an optimal policy from every state. Each constituent MDP requires $\Omega(T^2A)$ to find an optimal policy and there are $\lfloor \frac{N}{T} \rfloor$ such MDPs. Therefore, $\Omega(NAT)$ timesteps are required. □

CHAPTER 9

Model Building and Exploration

In the last chapter, we saw that the maximum number of timesteps, while R_{max} is not executing a near-optimal policy, is $O(\frac{N^2AT^3}{\varepsilon^3}\log^2\frac{NA}{\delta})$. In obtaining this result, an ε-accuracy condition (in an l_1 sense) was imposed on the empirically estimated transition model. This is a very different demand than all other algorithms discussed in this thesis, where no explicit demands were made on obtaining an accurate transition model of our MDP. Recall in phased value iteration (in section 2.5), the algorithm assumes access to a generative model, and after observing $O(\frac{NAT^3}{\varepsilon^2}\log\frac{NAT}{\delta})$ transitions, the algorithm returns a near-optimal policy from every state. Importantly, the empirical transition model from this latter approach (which uses $O(NA)$ samples neglecting $\log$ and other factors) is, in general, a poor approximation to the true transition model (which takes N^2A parameters to specify). However, these samples are sufficient to reliably compute an ε near-optimal policy.

Furthermore, the accuracy condition (8.5.1) sufficient for the success of R_{max} did *not* explicitly require that an accurate transition model be constructed. This condition only demanded that R_{max} obtain near-optimal policies in the induced MDPs M_K using the approximate MDP $\hat{M}_K$.

This raises the important question of whether or not our demand to obtain an accurate transition model is too stringent. Recall that our lower bound (8.3.4) on the number of timesteps in which our algorithm is not near-optimal is $O(\frac{NAT}{\varepsilon}\log\frac{1}{\delta})$. If this bound is tight (which is unclear), then an accurate model is not required to make our optimality guarantee. We focus on this question in this chapter.

In the first section of this chapter, we examine the sample complexity of a *model based* approach. In this approach, a model $\hat{P}$ is constructed to approximate the transition model in an MDP M and this approximate model is used for planning purposes. First, the result of Kearns and Singh [1999] is summarized, which shows that a model based approach (using access to a generative model) has a comparable sample complexity to phased value iteration. Importantly, both have sample complexity that is $O(NA)$ (neglecting log and other factors). We then ask the question of how difficult is to use this model $\hat{P}$ to reliably obtain near-optimal policies in k different induced MDPs M_K. It is easy to show that the overhead required to compute near-optimal policies for these k different induced MDPs, *specified independently of the samples*, is cheap — an additional factor of $\log k$ samples is sufficient.

The next section then examines the implications of this result for R_{max}. Recall that R_{max} computes an optimal policy in at most N induced MDPs before all states are known (see theorem 8.3.3). Crucially, R_{max} chooses these induced MDPs in a manner that is *dependent* on the prior observations. If we demand accuracy on all possible induced MDPs *apriori*, our analysis leads to roughly the same sample complexity bound since there are

2^N such MDPs (and so $\log k = N$). Unfortunately, the analysis presented in this chapter does not close the gap between our lower and upper bound, though it does provide a different interpretation as to why exploration in the online setting is challenging.

9.1. The Parallel Sampler

Let us start by reviewing the analysis of Kearns and Singh [1999], which was concerned with comparing the sample complexity of model based approaches with that of the Q-learning algorithm of Watkins [1989], which does not explicitly build a model of the MDP.

It is convenient to define a parallel sample, as in Kearns and Singh [1999]. A **parallel sample** for an MDP M is a a set of transitions $(s,a) \to s'$ for every state-action (s,a), where $s' \sim P(\cdot|s,a)$ and P is the transition model in M. Clearly, a parallel sample can be obtained with NA calls to the generative model $G(M)$, with one call per state-action.

The ApproximateMDP algorithm (15) shows the obvious way to construct an empirical MDP $\hat{M}$ using the generative model. The algorithm constructs the model $\hat{P}$ using the empirical transition frequencies in the m parallel samples and the reward function $\hat{r}$ is identical to the one in M. Let us also assume that M is a T-epoch MDP and so $\hat{M}$ is also a T epoch MDP.

Algorithm 15 ApproximateMDP$(G(M), m)$

(1) Obtain m parallel samples using $G(M)$

(2) Construct a T epoch MDP $\hat{M}$ using

$$\begin{aligned} \hat{P}(s'|s,a) &= \frac{\#\text{ of times } (s,a) \to s'}{m} \\ \hat{r}(s,a) &= r(s,a) \end{aligned}$$

(3) Return $\hat{M}$

The following lemma is on the number of parallel samples m that are sufficient in order for the optimal policy in $\hat{M}$ to be near-optimal in $\hat{M}$. Since m is the number of samples obtained at *each* state-action, then mNA is the total number of calls to the generative model. The result is essentially identical to that presented in Kearns and Singh [1999] (except that we treat the undiscounted case and explicitly state the T dependence).

LEMMA 9.1.1. *Let M be a T-epoch MDP. If $\hat{M}$ is an MDP returned by ApproximateMDP with inputs $G(M)$ and m where*

$$m = O\left(\frac{T^4}{\varepsilon^2}\log\frac{NAT}{\delta}\right)$$

then, with probability greater than $1-\delta$, any policy $\hat{\pi}$ that is optimal in $\hat{M}$ satisfies, for all s and $t < T$

$$V_{\hat{\pi},t,M}(s) \geq V^*_{t,M}(s) - \varepsilon\,.$$

Importantly, note this $O(\log N)$ dependency (fixing other constants) means that $\hat{P}(\cdot|s,a)$ is highly sparse, since there are N entries in $P(\cdot|s,a)$. Hence, in general, $\hat{P}(\cdot|s,a)$ is a terrible approximation to $P(\cdot|s,a)$ under any sensible measure.

Note that the total number of transitions is a factor of T more than the "direct" phased value iteration algorithm in section 2.5. This is a slightly different result than the one obtained in Kearns and Singh [1999] where their model based algorithm required fewer samples than their phased Q-learning algorithm.[1] Here, the (non-stationary) phased value iteration algorithm tightens up the number of calls to the generative model (as discussed in section 2.5), so this implies, in our analysis, that the direct algorithm calls the generative model fewer times (though it is not clear if this dependency is realized in practice).

The proof is based on the one sketched in Kearns and Singh [1999].

PROOF. Assume the following expectations are accurate under $\hat{P}$ for all t, s, a

$$|E_{s' \sim P(\cdot|s,a)} \left[V^*_{t,M}(s')\right] - E_{s' \sim \hat{P}(\cdot|s,a)} \left[V^*_{t,M}(s')\right]| < \varepsilon \,. \tag{9.1.1}$$

We later determine an appropriate value of m such that this condition holds.

Using the notation that $Q^*_{t,M}(s,a)$ is the state-action value of an optimal policy in M, then the above condition implies

$$\begin{aligned}
|Q^*_{t-1,M}(s,a) - Q^*_{t-1,\hat{M}}(s,a)| &\leq |E_{s' \sim P(\cdot|s,a)} \left[V^*_{t,M}(s')\right] - E_{s' \sim \hat{P}(\cdot|s,a)} \left[V^*_{t,\hat{M}}(s')\right]| \\
&\leq |E_{s' \sim P(\cdot|s,a)} \left[V^*_{t,M}(s')\right] - E_{s' \sim \hat{P}(\cdot|s,a)} \left[V^*_{t,M}(s')\right]| \\
&\quad + |E_{s' \sim \hat{P}(\cdot|s,a)} \left[V^*_{t,M}(s')\right] - E_{s' \sim \hat{P}(\cdot|s,a)} \left[V^*_{t,\hat{M}}(s')\right]| \\
&\leq \varepsilon + \max_s |V^*_{t,M}(s) - V^*_{t,\hat{M}}(s)| \\
&\leq \varepsilon + \max_{s,a} |Q^*_{t,M}(s,a) - Q^*_{t,\hat{M}}(s,a)|
\end{aligned}$$

where the last step follows since $V^*_{t,M}(s) = \max_a Q^*_{t,M}(s,a)$.

By recursion, it follows that for all $t < T$, $||Q^*_{t,M} - Q^*_{t,\hat{M}}||_\infty \leq T\varepsilon$. It is straightforward to use this result and the performance difference lemma 5.2.1 to show that this implies the greedy policy $\hat{\pi}$ is $2T^2\varepsilon$ near-optimal. Hence, we make the replacement $\varepsilon \leftarrow \frac{\varepsilon}{2T^2}$.

Now we address the number of samples m that allows equation 9.1.1 to be satisfied with $\varepsilon \leftarrow \frac{\varepsilon}{2T^2}$. Since there are NAT constraints, by Hoeffding's bound and the union bound, the probability of an approximation error of more than $\frac{\varepsilon}{2T^2}$ is less than $NAT \exp(-\frac{\varepsilon^2}{2T^4} m)$. The result follows by demanding that this be less than δ and solving for m. □

As discussed in Kearns and Singh [1999], it is easy to show that if we desire to compute optimal policies in k MDPs which differ only in the their reward functions, then this can be done with the single model $\hat{P}$ with the overhead of obtaining only an additional factor of $O(\log k)$ samples — provided that reward functions are chosen independently of $\hat{P}$.

Instead, let us consider the case in which we wish to find optimal policies in the induced MDPs $M_{K_1}, M_{K_2} \ldots M_{K_k}$, where $K_1, K_2, \ldots K_k$ are k sets of states. Based on the single MDP $\hat{M}$, we also have a set of induced MDPs $\hat{M}_{K_1}, \ldots \hat{M}_{K_k}$. Let us consider planning with respect to these induced MDPs in an attempt to obtain a near-optimal policy for each M_{K_i}. The following theorem is on the number of transitions required to obtain accurate

[1] Kearns and Singh [1999] did not explicitly examine the $\frac{1}{1-\gamma}$ dependence in the discounted case, but the different ε dependence in their results is directly related to the different $\frac{1}{1-\gamma}$ dependence, had this factor been included.

policies $\hat{\pi}_i$ for all M_{K_i}. As expected, there is only an additional $\log k$ cost. Crucially, in order to obtain this $\log k$ dependence the sets K_i must be chosen independently of $\hat{M}$.

THEOREM 9.1.2. *(Multiple Induced MDPs) Let M be a T-epoch MDP. If $\hat{M}$ is an MDP returned by ApproximateMDP with inputs $G(M)$ and m where*

$$m = O\left(\frac{T^4}{\varepsilon^2}(\log k + \log\frac{NAT}{\delta})\right)$$

and if $K_1, K_2, \ldots K_k$ are sets of states chosen independently of the transition model in $\hat{M}$, then, with probability greater than $1-\delta$, then any policy $\hat{\pi}_i$ that is optimal in $\hat{M}_{K_i}$ satisfies, for all s, K_i, and $t < T$:

$$V_{\hat{\pi}_i,t,M_{K_i}}(s) \geq V^*_{t,M_{K_i}}(s) - \varepsilon\,.$$

PROOF. By construction, the MDP $\hat{M}_{K_i}$ is distributed identically to one that had been created using a generative model for M_{K_i}. This is because the parallel samples contain no dependencies between samples and the choice of K_i does not correlate the samples with transition model in $\hat{M}_{K_i}$. Hence, it suffices to demand an error less than $\frac{\delta}{k}$ for each M_{K_i}, and the previous lemma implies the result. □

9.2. Revisiting Exploration

Recall from lemma 8.5.6 that the number of times that R_{max} tries each state-action before the state becomes known is

$$m = O\left(\frac{NT^2}{\varepsilon^2}\log\frac{NA}{\delta}\right)\,. \tag{9.2.1}$$

With this value of m, we have the strong guarantee that all policies in $\hat{M}_K$ are accurate estimates of their values in M_K (with error probability less than δ). In obtaining this result, we demanded that each $\hat{P}(\cdot|s,a)$ be an accurate approximation to $P(\cdot|s,a)$ in an l_1 sense (see definition 8.5.4).

Recall that this l_1 condition is a more stringent condition than that required by our accuracy condition 8.5.1 which only demanded that the optimal policies derived from $\hat{M}_K$ be near-optimal policies in M_K in order for R_{max} to succeed.

Importantly, note that this value of m is a factor of N more than that provided in lemma 9.1.1, which was only concerned with obtaining a single optimal policy using a generative model (though the T dependence is less). This lemma shows that a highly impoverished model of the MDP is sufficient to compute a near-optimal policy. The question we now investigate is: *is this l_1 condition on $\hat{P}$ too stringent?* Our lower bound (theorem 8.3.4) leaves open this possibility.

Recall that R_{max} uses at most N induced MDPs since the known set is only increasing in size. Hence, there are at most N induced MDPs for which we desire near-optimal policies. The proof for the last theorem implied that if these induced MDPs were chosen *independently* of the observed samples then only $O\left(\frac{T^4}{\varepsilon^2}\log\frac{NAT}{\delta}\right)$ visits per state-action are sufficient to satisfy our accuracy condition for R_{max} (since there is only a $\log k = \log N$ overhead).

Crucially, as we argue in the next subsection, these $\hat{M}_K$ are chosen *dependently* on the observed samples so we cannot invoke the last theorem to enjoy this minor $O(\log N)$

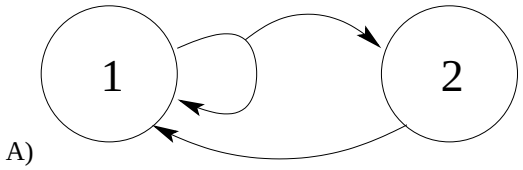

FIGURE 9.2.1. See text for description

overhead. In light of this dependency, we then consider demanding that we obtain accurate optimal policies on *all possible* induced MDPs that R_{max} might encounter. This would ensure the success of R_{max}. Unfortunately, there are 2^N such MDPs.

9.2.1. Dependencies on an L-Path. Let c be an L-path sampled according to the distribution $\Pr(\cdot|R_{max}, M, s_0)$. Clearly, each transition $(s,a) \to s'$ is generated according to the transition model in M. Also, each $\hat{M}_K$ that is used at time t is constructed using only the *number* of times in which each state action (s,a) was visited in the subpath c_t. Hence, R_{max} does not actually need to observe complete transitions $(s,a) \to s'$ when it constructs $\hat{M}_K$. Nonetheless, the following example demonstrates that this construction could correlate K with the empirical transition matrix of $\hat{M}_K$.

EXAMPLE 9.2.1. Consider a two state MDP M shown in figure 9.2.1, where there is one action at each state. The transition from 1 has equal probability to go to either state, *ie* $P(2|1) = \frac{1}{2}$ and the transition from 2 is to state 1. Conditioned on the occurrence of m_1 and m_2 visits to state 1 and state 2, we can infer *exactly* what the empirical probability of $1 \to 2$ in this chain and it is $\frac{m_1}{m_1+m_2}$. Hence, any decision made based on only m_1 and m_2 is equivalent to making a decision using the empirical frequencies in this path.

The previous example can be viewed as a problem of "mixing". If independence samples are desired, a clever algorithm might only accept samples from state-actions if there is a sufficiently long time between the last accepted sample, such that each sample is independent. Standard definitions of the "mixing time" formalize this notion. In Kearns and Singh [1999], this suggestion is discussed as a method to simulate the parallel sampler.

Unfortunately, this method is, in general, only applicable to stationary algorithms (*ie* stationary policies), since paths provided by these algorithms satisfy the Markov property. A non-stationary algorithm could impose arbitrarily long dependencies in the path such that no amount of "mixing" could provide near *i.i.d.* samples. Essentially, the algorithm's choice of action at time t could be dependent on the state-action at time $t = 0$, and this violates the Markov property. In general, algorithms required for exploration are non-stationary, since based on the prior experience the algorithm decides to explore or exploit.

9.2.2. Independence and R_{max}. This section addresses the number of samples sufficient to guarantee that all computed escape policies are accurate.

First, let us review the setting. The case considered is one in which a path c is obtained using some algorithm $\mathcal{A}$ (say R_{max}). At each point in time $t \leq L$, we could consider building some $\hat{M}_K$ where K is the m-known set on the subpath $c_t = (s_0, a_0, \cdots, s_t)$. The question of interest is: how large does m need to be such that all $\hat{M}_K$'s constructed from some subpath c_t are accurate enough to provide near-optimal policies for the corresponding M_K? This is precisely what is required for the success of R_{max}.

The crucial problem is that the induced MDP must be chosen *independently* of the samples in order to apply the previous theorem. One way to cope is to demand accuracy on *all possible* induced MDPs that R_{max} may encounter. Unfortunately, there could be 2^N such induced MDPs, which is due to the fact that each induced MDP corresponds to a subset K of the state space and there are 2^N such subsets.

The following theorem addresses the aforementioned sample size by appealing to our previous theorem on multiple induced MDPs (9.1.2) with $k = 2^N$. Although we are dealing with paths rather than parallel samples the proof shows that we can use previous theorem if we set $k = 2^N$.

THEOREM 9.2.2. *Let $\mathcal{A}$ be an algorithm and s_0 be a state with respect to an L-epoch MDP M, Let c be an L-path sampled from $Pr(\cdot|\mathcal{A}, M, s_0)$. If*

$$m = O\left(\frac{T^4}{\varepsilon^2}(N + \log\frac{NAT}{\delta})\right) .$$

then with probability greater than $1 - \delta$, for all times $\tau \leq L$, if $\hat{M}_K$ is an induced MDP with respect to the m-known set on subpath c_τ, then any optimal policy $\hat{\pi}$ that is optimal on $\hat{M}_K$ satisfies for all s and $t < T$

$$U_{\hat{\pi},t,M_K}(s) \geq U^*_{t,M_K}(s) - \varepsilon .$$

Here, we use U because technically M_K is an L-epoch MDP, so U denotes that the value functions are with respect to T (see section 8.4.1).

Unfortunately (and not surprisingly), this analysis does reduce the gap between our lower and upper bound, since we are demanding accuracy on 2^N MDPs. Also note the T dependence is slightly worse in this analysis, so it is not worth carrying this result through to provide an alternative upper bound for the sample complexity of exploration (compare to equation 9.2.1).

The proof just requires a careful (yet intuitive) argument that we can use our last theorem when we construct our model from an L-path rather than using parallel samples.

PROOF. Consider generating m parallel samples, where $m = O\left(\frac{T^4}{\varepsilon^2}(N + \log\frac{NAT}{\delta})\right)$. For this set of parallel samples, the last theorem implies the corresponding MDP $\hat{M}$ is one in which the optimal policies in all $k = 2^N$ induced MDPs are ε near-optimal in their corresponding exact induced MDP, with probability of error less than δ.

Now let us consider a generative procedure for sampling c from $\Pr(\cdot|\mathcal{A}, M, s_0)$. This procedure uses the same m parallel samples as mentioned above to generate transitions (without reuse) as follows. If $\mathcal{A}$ takes action a at state s, then we use a transition $(s, a) \to s'$ (without reuse) from the parallel sample to set the next state in the path to s'. If all the samples at (s, a) have been used in the parallel sample, then we just generate a next state from $P(\cdot|s, a)$. It should be clear that this procedure provides a path c that is sampled according to $\Pr(\cdot|\mathcal{A}, M, s_0)$. By the previous argument, all induced MDPs constructed from this path must be accurate with probability of error less than δ, since these induced MDPs are just a subset of all possible induced MDPs (constructed from the m parallel samples). □

CHAPTER 10

Discussion

This thesis has summarized recent progress toward obtaining efficient algorithms and has provided novel algorithms with strong performance guarantees. We now discuss some of the important issues and insights raised by this work.

10.1. N, A, and T

A critical issue for the application of reinforcement learning algorithms to realistic problems is how the sample (and computational) complexity scales with N, A, and T. The first two subsections examine the sample complexity connections between N and T. Perhaps the most important practical contribution of this work is in understanding the tradeoffs made in order to provide algorithms which have no dependence on N and polynomial dependence on T. The final subsection points out why the scaling with A is fundamentally different.

As a unifying theme to this section, let us compare the algorithms to a degenerate MDP with $T = 1$ and $A = 2$. For this case, let us assume we are interested in a good performance under the initial state distribution D, *ie* we desire to find a good policy as measured by the function $E_{s\sim D}[V_\pi(s)]$. This makes the reinforcement learning problem related to a classification problem under the standard indicator loss function (with the caveat that the rewards weight the indicator loss, see subsection 6.4.1). This is a minor variant of perhaps the best studied problem in machine learning, where almost all theoretical guarantees have no dependence on the size (or dimensionality) of the input domain, which is N here. Now let us ask ourselves: why does the problem become so difficult as we increase these parameters?

10.1.1. N vs. A^T Dependence. Let us start with value function methods. Sample based algorithms such as phased value iteration have a polynomial sample complexity in both N and T (ignoring other relevant factors) when we assume access to a generative model (see section 2.5). For $T = 1$, the phased value iteration just learns the reward function at *every* state in the input domain, and the algorithm performs *perfectly* for any choice of D (recall the reward function is deterministic).

For approximate value function methods (as in chapter 3), performance bounds are stated in terms of max norm regression errors, which suggests that there is no relevant measure over the state space with respect to which it is appropriate to optimize. Loosely, the reason the max norm error is relevant for a greedy (stationary) policy update from a policy π to a policy π' is because the greedy policy π' may visit only those states where the worst case errors have occurred — compounding this error over the entire horizon. Furthermore, the reason it visits these states might be due to the errors themselves.

It is this max norm error that is often the thorn in providing strong sample complexity results that are independent of N in an approximate setting. Also, note the incongruous situation: when $T = 1$, it is clear that D is the appropriate measure to use and that max norm error bounds over the state space are *not* relevant.

Now let us consider the "tree based" methods of Kearns, Mansour, and Ng [1999,2000], starting with the sparse sampling algorithm which can be viewed as the exact counterpart to an algorithm such as phased value iteration (see section 2.5). Here, the algorithm pays a runtime sample complexity that is exponential in T, but has no N dependence. For the degenerate $T = 1$ case, the algorithm also behaves perfectly, but here the algorithm only calls the generative model at runtime to do the classification.

The corresponding approximate algorithm to the sparse sampling algorithm is effectively the trajectory tree method. Here, the key idea is that a single tree of size $O(A^T)$ simultaneously provides an unbiased estimates for any policy. As in supervised learning, this method has no dependence on N (for both $T \neq 1$ and $T = 1$) and a linear dependence on the complexity of the policy (or hypothesis) class. Furthermore the algorithm always uses the measure D when sampling the root state[1] (whereas for approximate value functions methods, only for $T = 1$ is it clear to use a measure). This method provides a more parsimonious generalization of supervised learning to reinforcement learning, and we return to this point in the next section.

10.1.2. A^T vs. μ Dependence. In practical applications, we often are dealing with both large state spaces and large horizon times. Clearly, we desire algorithms that can cope with this situation. However, as argued, we must expect a tradeoff to be made, since an $O(N)$ or $O(A^T)$ sample complexity dependence is intrinsic to the problem (see the lower bounds in section 2.5).

The "mismeasure" results in chapter 5 showed how the difference between two policies can be stated in terms of the advantages of one policy and the future state distribution of the comparison policy. Here, we can view the comparison policy as providing the "test" distribution, and to compete favorably against this policy, our advantages must be small on average with respect to this "test" distribution. This sheds some light on the difficulty of the reinforcement learning problem — we may be ignorant of what this "test" distribution is.

Of course neither approximate value function methods nor the trajectory tree methods take this notion of a "test" distribution into account, and their more general guarantees come at a cost. The approximate value function methods use the max norm condition which bounds the performance under any "test" distribution. The trajectory tree method effectively obtains samples from all possible "test" distributions (which scales as $O(A^T)$).

It is likely that to make reinforcement learning work on any real and challenging domain significant problem dependent information must be taken into account. We have used this idea of a "test" distribution as a way of incorporating domain knowledge — effectively building in expectations where a good policy visits. If we have such information, then it is clear we would like to bias our optimization procedure to favor these states.

One natural approach is to force the advantages to be small under a measure μ of our choosing. The hope was that since by making a less stringent optimality demand, then we

[1] In our presentation, we considered building the trees from a single state s_0. However, in general, if D is the input distribution, then the trees would start with a root state $s \sim D$.

can find algorithms with a lower sample complexity. Example 3.2.4 suggested that it is not reasonable to expect standard greedy value function approaches to achieve this small advantage condition and chapter 4 suggested that it is unclear as how to force gradient methods to achieve this condition either.

Two algorithms were presented, μ-PolicySearch and CPI, which have sample complexity bounds that are independent of N and polynomial in T. Importantly, these methods still preserve the linear dependence on the complexity of a restricted policy class Π. The trade-off is that these algorithms only guarantee to return policies which have advantages that are small with respect to μ. However, we can still make a non-trivial optimality guarantee based on μ (as in theorems 6.3.1 and 7.3.1).[2] We return to the reasons for their success in the next section.

Another promising approach is the approximate linear programming approach of de Farias and Van Roy [2001]. This method guarantees that a "good" approximation to the optimal value function can be obtained, where "good" means that the *average* approximation error with respect to μ is small relative to all other functions in the (linear) function approximating class (see section 3.3). Again, this method stresses the importance in the choice of the measure μ. de Farias and Van Roy [2001] also provide conditions under which sample complexity bounds can be obtained that are independent of the size of the state space and that depend polynomially on the number of features. However, the nature of the μ-guarantee in the linear programming approach is different from μ-PolicySearch and CPI — the guarantee is with respect to the error in approximate value function rather than the advantages of the output policy itself (see section 3.3).

10.1.3. A Dependence. We have made no effort to deal with the polynomial A dependence in this work. Furthermore, *all* the methods that we have reviewed and presented in this thesis have difficulty in dealing with large action spaces.[3]

Again, let us consider the $T = 1$ case, but with $A = \infty$. If the action space is uncountable, this problem is similar to a regression problem, where typically additional Lipschitz continuity conditions are assumed in order to make sensible performance guarantees.

Since in the supervised learning case ($T = 1$), additional assumptions are usually made to deal with the infinite action case, then it is likely as we scale T we must also consider additional assumptions to deal with this case. Lipschitz continuity conditions for $T \neq 1$ have been consider by Ng and Jordan [2000] and it should be possible to apply these conditions to our μ based algorithms.

10.2. From Supervised to Reinforcement Learning

Though reinforcement learning is a more general problem than supervised learning, much insight has been gained by considering how we can apply techniques from supervised learning to reinforcement learning.

[2] As discussed in section 6.3.1, there are no more informative and tighter bounds on this relatively common case in which we know our error is small under one measure and we are interested in the error under a different measure. However, as in supervised learning, we have the potential to compete favorably even for cases in which the test distribution differs significantly from μ. Intuitively, we are lucky if the test distribution doesn't focus on states where there are large advantages.

[3] Recall the discussion in section 4.2.2 where we argued that gradient methods also have difficulty with large action spaces. This is due to variance reasons related to importance sampling. Essentially, the natural form of the gradient involves a *sum* over actions while it involves an *expectation* over the state space.

10.2.1. The Trajectory Tree Method and Beyond. Kearns, Mansour, and Ng showed how standard complexity ideas from supervised learning can be applied to the reinforcement learning setting to obtain uniform convergence results for all policies within some restricted "hypothesis" set of policies Π. Here, the bound on the sample complexity is linear in the complexity of Π (and of course exponential in T but independent on N). This method replicates a fundamental property in supervised learning, that a tree can provide simultaneous feedback on all policies in the hypothesis set Π.

Clearly, building trees of size $O(A^T)$ is not feasible and we desire polynomial T dependence for practical approaches.[4] However, when we move to a more practical setting, we do not wish to lose this efficient reuse of samples. Gradient methods are one method to potentially avoid building trees. As a heuristic, these methods could be used with a measure μ in an attempt to alleviate their exploration related problems (see chapter 4). However, it is not clear how they hang on to this idea of efficient reuse (see 6.4.3). Here, we have argued that the μ-PolicySearch and CPI algorithms are natural methods which both efficiently reuse samples and optimize with respect to a measure μ. Let us now discuss what in particular allowed these algorithms to achieve both of these goals.

10.2.2. Successful Policy Update Rules. Both μ-PolicySearch and CPI algorithms use PolicyChooser subroutines which attempt to return decision rules $\pi' \in \Pi$ which choose actions with "large" advantages with respect to the current policy π, and the notion of "large" is an *average* one based on the measure μ. These PolicyChooser algorithms efficiently reuse samples to find a "good" π' in a manner similar to the trajectory tree method (except now the measure μ is used in lieu of building the tree, see subsection 6.3.3). A central theme in part 2 was the construction of policy update rules which drive the advantages to be small with respect to a measure μ using the output decision rules of the PolicyChooser.

The ingredients for successful updates for both μ-PolicySearch and CPI are twofold. First, both algorithms make "small" policy changes. Second, both algorithms are variants of policy iteration. This means that each subsequent decision rule attempts to choose better actions by taking into account the advantages of the current policy.

In μ-PolicySearch, the small policy change is implemented by only altering the policy at one decision epoch at a time starting from time $T-1$ and working down to time 0. The policy iteration nature of the algorithm forces the PolicyChooser to construct the decision rule $\pi(\cdot, t)$ by taking into account the remaining sequence of decision rules $\pi(\cdot, t+1), \ldots, \pi(\cdot, T-1)$. This allows max norm error bounds to be avoided (such as in the regression version of non-stationary approximate policy iteration, see section 5.3). The final policy returned by μ-PolicySearch is both deterministic and non-stationary (assuming that Π is a class of deterministic decision rules).

In contrast, CPI returns a good *stationary* policy. The natural update rule implemented by CPI just mixes the new policy with the old policy using some mixing parameter α (see equation 7.2.1). Unlike in μ-PolicySearch which halts after T updates, it was much harder

[4]The PEGASUS method of Ng and Jordan [2000] does not need to explicitly build the tree of size A^T. Here, we can view fixing the seed to the random number generator as proving a compact representation of the tree, and the relevant question is now a computational one: how many transitions must be computed using this method? We argue this could, in some cases, be exponential in T (see 6.1.3). Essentially, PEGASUS is a variance reduction method and does not address the problem of exploration. PEGASUS could also be used with the μ based policy search methods for reducing variance.

to understand the behavior of this update rule and we had to think more carefully about when to halt CPI and how to set α.

Note that both algorithms are using all their previous decision rules — μ-Policy search is executing all the decision rules in backward order of construction while CPI is mixing between all its decision rules (in order to preserve stationarity).

Interestingly, we have only presented μ-based algorithms (with polynomial T dependence) which output either *stochastic, stationary* policies or *deterministic, non-stationary* policies. It is not clear how to present an algorithm which has a similar μ-based guarantee (with respect to the advantages) and that outputs a *deterministic and stationary* policy.

10.2.3. Query Learning. A fruitful direction to consider is query (or active) learning (as in Angluin [1987]). The typical setting is one in which the learner is permitted to actively query the instances over an input space in order to reduce generalization error with respect to a fixed distribution D. This setting has been shown to help reduce the generalization error with respect to D in a variety of problems. Ideally, we would like to consider an algorithm which is not tied to using a single measure D and perhaps tries to efficiently and robustly reduce the error with respect to multiple measures. Though, in the extreme, this leads back to dealing with the max norm error or the A^T dependence. This direction for future work might provide a more general means to tackle the exploration problem rather than using a fixed distribution μ.

10.3. POMDPs

We have so far tended to avoid issues of planning and exploration in partially observable Markov decision processes (POMDPs).

10.3.1. Planning. Although the computational complexity of exact planning in MDPs and POMDPs is different (see Littman [1996]), there is a close connection between approximate planning in MDPs and POMDPs. Intuitively, the reason is that using only partial information for approximate planning in an MDP can often be viewed as working in a POMDP framework. For this reason, gradient methods have direct applicability to POMDPs.

The trajectory tree method was originally presented as means for sample-based planning in POMDPs. Our summary only described this algorithm for MDPs, but it is clear that a single tree in a POMDP provides simultaneous estimates for the values of all policies (and so the same uniform convergence arguments can be applied in the POMDP setting). Of course the policy class Π must be restricted to use only observable information.

In our setting, it is not too difficult to see that our μ-based planning approaches can also be applied to POMDPs. However, now μ is a distribution over history vectors or belief states, and, of course, the policy class is only restricted to use observable information. Here the problem of choosing and representing a good μ becomes more interesting and challenging (and we could certainly consider using "memoryless" μ's).

10.3.2. Bayesian "Exploration" in MDPs. In the sensible Bayesian view of reinforcement learning, the agent has a prior Q over MDPs and the goal is maximize some measure of the future reward when the agent is placed in an MDP $M \sim Q$. As discussed in chapter 8, this solution has a well defined optimal (memory dependent) policy, which makes an *optimal* exploration/exploitation tradeoff.[5] In fact, the notions of exploration and exploitation are rather artificial in this setting.

It is clear that this problem can be cast in a POMDP, where the agent does not have knowledge of which MDP it is in. Since Q is known, the problem is a purely computational one. Hence, one could attempt to obtain an approximate solution using the trajectory tree method or the μ-based planning algorithms (assuming that we can sample from Q and M efficiently). This may be a promising approach for "exploration" in large state spaces.

The appeal of this framework is that a sensible notion of optimality is well defined. The drawback is that it is often not clear how to construct a prior that is indicative of the task at hand and the results may be quite sensitive to the prior distribution used.

10.4. The Complexity of Reinforcement Learning

The overall complexity of reinforcement learning can be considered to be both the *sample* and *computational* complexity required to achieve "learning" given only some sampling model of the environment and a performance criterion. Much work has gone into understanding the computational complexity of exactly solving MDPs and POMDPs when they are specified in a tabular representation (see Littman [1996] for review).

Once we no longer assume complete knowledge of the MDP or desire approximate algorithms, there is a host of new complexity related questions, which we have discussed in this thesis. For instance, in the policy search setting we are interested in the overall complexity of finding a good policy within a restricted policy class Π.

As in many supervised learning analyses, we have only considered using an arbitrary policy class Π in our policy search setting and have not considered how to efficiently manipulate this class Π. The bounds in part 2 are purely information theoretic. An important direction is in understanding the computational complexity of performing the necessary optimizations using this policy class.

For the trajectory tree method, this optimization may be an expensive proposition due to the sheer size of the trees (though the performance guarantees are unrestricted). For the μ-based methods, we have pointed out that the optimization is equivalent to minimizing a cost sensitive classification loss function. This might open a door to the application of more standard supervised learning methods in which the computational complexity and optimization tools are more well-studied.

Furthermore, not only are we interested in being able to efficiently optimize within our policy class, we are also interested in constructing policy classes that contain good policies. Understanding, how to use domain knowledge (which often comes in the knowledge of environment dynamics) to construct good policy classes is important for the design of practically successful algorithms. Unfortunately, it should be noted that for factored dynamics it may not be possible construct compact policy classes (Allender, Arora, Moore,

[5] In this case, the value of an algorithm is just its expected return in an MDP M that is sampled according to Q, and the expectation is taken with respect to Q and the $M \sim Q$. The optimal algorithm is the one which maximizes this value.

Kearns, and Russell [2003]). A direction we have not considered and that is a relatively open area is the use of non-parametric policy search methods which may help to avoid the need for making strong parametric assumptions which are often violated in practice.

Bibliography

[1] Allender, E., Arora, S., Moore, C., Kearns, M., and Russell, A. (1993). A Note on the Representational Incompatibility of Function Approximation and Factored Dynamics. To appear in: *Proceedings of NIPS.*

[2] Angluin, D. (1987). Queries and concept learning. *Machine Learning*, 2:319-432.

[3] Anthony, M. and Bartlett, P.L. (1999). *Neural Network Learning: Theoretical Foundations.* Cambridge University Press.

[4] Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10.

[5] Bagnell, J. and Schneider J. (2001). Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. *Proceedings of the International Conference on Robotics and Automation, IEEE*.

[6] Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transaction on Systems, Man and Cybernetics.

[7] Brafman, R. I. and Tennenholtz, M. (2001). R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. In *Proceedings of the Eighteenth International Joint Conferences on Artificial Intelligence.*

[8] Baird, L. C. (1993). Advantage updating. Technical report. WL-TR-93-1146, Wright-Patterson Air Force Base.

[9] Baird, L. C. (1995). Residual algorithms : Reinforcement learning with function approximation. In *Machine Learning : proceedings of the Twelfth International Conference*.

[10] Baird, L. C. and Moore, A. (1999). Gradient descent for general reinforcement learning. In *Neural Information Processing Systems*, 11.

[11] Bartlett, P. and Baxter, J. (2000). Estimation and approximation bounds for gradient-based reinforcement learning. Technical report. Australian National University.

[12] Baxter, J. and Bartlett, P. (2001). Infinite-Horizon Policy-Gradient Estimation. *Journal of Artificial Intelligence Research*, 15.

[13] Baxter, J., Tridgell, A., and Weaver, L. (2000). Learning to Play Chess Using Temporal-Differences. *Machine Learning*, 40.

[14] Bellman, R. E. (1957). *Dynamic programming*, Princeton University Press, Princeton, NJ.

[15] Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, NJ.

[16] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

[17] Boyan, J. A. and Moore, A. W. (1995). Generalization in reinforcement learning: safely approximating the value function. In *Advances in Neural Information Processing Systems* 6.

[18] de Farias, D. P. and Van Roy, B. (2001). On Constraint Sampling in the Linear Programming Approach to Approximate Dynamic Programming. Operations Research (submitted 2001).

[19] de Farias, D. P. and Van Roy, B. (2001). The Linear Programming Approach to Approximate Dynamic Programming. Operations Research (submitted 2001).

[20] Fiechter, C. (1994). Efficient reinforcement learning. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*. ACM Press.

[21] Gittins, J. C. (1989). *Multi-armed Bandit Allocation Indices*. Wiley-Interscience series in systems and optimization.

[22] Glynn, P. W. (1986). Stochastic approximation for Monte Carlo optimization. In *Proceedings of the 1986 Winter Simulation Conference*.

[23] Gordon, G. J. (1999). Approximate Solutions to Markov Decision Processes. PhD thesis, Carnegie Mellon University.

[24] Gordon, G. J. (1996). Chattering in SARSA(λ) - A CMU Learning Lab Internal Report.

[25] Gordon, G. J. (2001). Reinforcement learning with function approximation converges to a region. *Advances in Neural Information Processing Systems.*

[26] Gordon, G. J. (1995). Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning.*

[27] Haussler, D. (1992). Decision theoretic generations of the PAC-model for neural nets and other applications. *Information and Computation*, 100, 78–150.

[28] Kakade, S. (2001). Optimizing Average Reward Using Discounted Rewards. In *Proceedings of the 14th Annual Conference on Computational Learning Theory*.

[29] Kakade, S. (2002). A Natural Policy Gradient. In *Advances in Neural Information Processing Systems*, 14.

[30] Kakade, S. and Langford, J. (2002). Approximately Optimal Approximate Reinforcement Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning.*

[31] Kearns, M., and Koller, D. (1999). Efficient Reinforcement Learning in Factored MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence.*

[32] Kearns, M., Mansour, Y. and Ng, A. (1999). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence.*

[33] Kearns, M., Mansour, Y. and Ng, A.Y. (2000). Approximate planning in large POMDPs via reusable trajectories.In *Neural Information Processing Systems* 12. MIT Press.

[34] Kearns, M., and Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. In *Proceedings of the Fifteenth International Conference on Machine Learning.*

[35] Kearns, M. and Singh, S. (1999). Finite sample convergence rates for Q-learning and indirect algorithms. In *Neural Information Processing Systems* 12. MIT Press.

[36] Kearns, M., Schapire, R., and Sellie, L. (1994). Toward efficient agnostic learning. *Machine Learning*, 17(2/3):115–142.

[37] Kearns, M. and Vazirani, U. (1994). *An introduction to computational learning theory*. MIT Press, Cambridge, MA.

[38] Kimura, H., Yamamura, M., and Kobayashi, S. (1995). Reinforcement Learning by Stochastic Hill Climbing on Discounted Reward. In *Proceedings of the 12th International Conference on Machine Learning*.

[39] Koenig, S. and Simmons, R. G. (1993). Complexity Analysis of Real-Time Reinforcement Learning. In *Proceedings of the International Conference on Artificial Intelligence.*

[40] Konda, V. and Tsitsiklis, J. (2000). Actor-Critic Algorithms. In *Advances in Neural Information Processing Systems*, 12.

[41] Langford, J. Zinkevich, M. & Kakade, S. (2002). Competitive Analysis of the Explore/Exploit Tradeoff. In *Proceedings of the Nineteenth International Conference on Machine Learning*.

[42] Littman, M. L., Dean, T. L. and Kaelbling, L.P. (1995). On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh International Conference on Uncertainty in Artificial Intelligence.*

[43] Littman, M. L. (1996). Algorithms for Sequential Decision Making. Ph.D. dissertation. Brown University, Department of Computer Science, Providence, RI.

[44] Marbach, P. and Tsitsiklis, J. N. (2001). Simulation-Based Optimization of Markov Reward Processes. *IEEE Transactions on Automatic Control*, Vol. 46, No. 2, pp. 191-209.

[45] Meuleau, N., Peshkin, L., and Kim, K. (2001). Exploration in Gradient-Based Reinforcement Learning. Technical report. Massachusetts Institute of Technology.

[46] Ng, A. Y. and Jordan, M (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence, Proceedings of the Sixteenth Conference.*

[47] Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3).

[48] Peshkin, L., Meuleau, N., Kim, K. and Kaelbling, L. (1999). Learning policies with external memory. In *Proceedings of the Sixteenth International Conference on Machine Learning*.

[49] Precup, D., Sutton, R.S., and Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. *Proceedings of the 18th International Conference on Machine Learning.*

[50] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York.

[51] Singh, S. (1994). Learning to Solve Markovian Decision Processes. PhD thesis, University of Massachusetts.

[52] Singh, S., and Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Neural Information Processing Systems*, 9.

[53] Singh, S., Jaakkola, T., and Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings 11th International Conference on Machine Learning*.

[54] Singh, S. and Yee, R. C. (1994). An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16:227.

[55] Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* MIT Press.

[56] Sutton, R.S., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Neural Information Processing Systems*, 13. MIT Press.

[57] Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6.

[58] Thrun, S. B. (1992). Efficient Exploration in Reinforcement Learning. Technical report. Carnegie Mellon University.

[59] Tsitsiklis, J. N. and Van Roy, B. (1997). An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, Vol. 42, No. 5.

[60] Valiant, L.G. (1984). A Theory of the Learnable. Communications of the ACM 27, pp. 1134-1142.

[61] Vapnik, V.N. (1982). *Estimation of dependences based on empirical data.* Springer-Verlag, New York.

[62] Watkins, C. J. C. H. (1989). Learning from Delayed Rewards. PhD thesis, Cambridge University.

[63] Weaver, L. and Baxter, J. (1999). Reinforcement Learning From State Differences. Technical report. Australian National University.

[64] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8:229-256.

[65] Williams, R. J., and Baird, L. C. (1993). Tight Performance Bounds on Greedy Policies Based on Imperfect Value Functions. Technical report. Northeastern University.

[66] Williams, R. J., and Baird, L. C. (1993). Analysis of Some Incremental Variants of Policy Iteration: First Steps Toward Understanding Actor-Critic Learning Systems. Technical report. Northeastern University.

[67] Whitehead, S. D. (1991). A Study of Cooperative Mechanisms for Faster Reinforcement Learning. Technical report. University of Rochester.

[68] Zhang, W. and Dietterich, T. (1995) A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*.