A UNIFYING FRAMEWORK FOR COMPUTATIONAL REINFORCEMENT LEARNING THEORY

BY LIHONG LI

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Michael L. Littman

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2009

ABSTRACT OF THE DISSERTATION

A Unifying Framework for Computational Reinforcement Learning Theory

by Lihong Li

Dissertation Director: Michael L. Littman

Computational learning theory studies mathematical models that allow one to formally analyze and compare the performance of supervised-learning algorithms such as their sample complexity. While existing models such as *PAC* (*Probably Approximately Correct*) have played an influential role in understanding the nature of supervised learning, they have not been as successful in reinforcement learning (RL). Here, the fundamental barrier is the need for active exploration in sequential decision problems.

An RL agent tries to maximize *long-term* utility by *exploiting* its knowledge about the problem, but this knowledge has to be acquired by the agent itself through *exploring* the problem that may reduce *short-term* utility. The need for active exploration is common in many problems in daily life, engineering, and sciences. For example, a Backgammon program strives to take good moves to maximize the probability of winning a game, but sometimes it may try novel and possibly harmful moves to discover how the opponent reacts in the hope of discovering a better game-playing strategy. It has been known since the early days of RL that a good tradeoff between exploration and exploitation is critical for the agent to learn fast (*i.e.*, to reach near-optimal strategies with a small *sample complexity*), but a general theoretical analysis of this tradeoff remained open until recently.

In this dissertation, we introduce a novel computational learning model called *KWIK* (*Knows What It Knows*) that is designed particularly for its utility in analyzing learning problems like RL where active exploration can impact the training data the learner is exposed to. My thesis is that the KWIK learning model provides a flexible, modularized, and unifying way for creating and analyzing reinforcement-learning algorithms with provably efficient exploration. In particular, we show how the KWIK perspective can be used to unify the analysis of existing RL algorithms with polynomial sample complexity. It also facilitates the development of new algorithms with smaller sample complexity, which have demonstrated empirically faster learning speed in real-world problems. Furthermore, we provide an improved, matching sample complexity lower bound, which suggests the optimality (in a sense) of one of the KWIK-based algorithms known as *delayed Q-learning*.

Acknowledgements

This dissertation could not have been completed without support and encouragement from many people. My greatest gratitude is extended to:

- My advisor, Michael Littman, the best mentor I could have. My academic pursuit would have been much harder without the wise guidance and plenty of freedom Michael gave. His helps are all-round, from spiritual encouragements to research methodology to writing styles and to to presenting skills. All I learned from him will be a treasure in my future career. Having Michael as my advisor was one of the luckiest things in my life.
- My M.Sc. co-advisors, Vadim Bulitko and Russ Greiner, from the University of Alberta, who first trained me to do research and supervised my earliest work on reinforcement learning. I am also grateful to Rich Sutton for teaching me reinforcement learning in his graduate class, which apparently aroused my interests in this field and prepared me well for further research.
- My thesis committee members: Michael Littman, Mike Pazzani, Rob Schapire, and Mario Szegedy, whose time and efforts in reading an earlier draft of this dissertation are appreciated. The document benefited a lot from their constructive feedback. In particular, Michael gave numerous suggestions that greatly improved the presentation, and Rob made an interesting observation that led to a simplified algorithm and analysis in §5.2.4.
- My qualifying exam committee members: Michael Littman, Joe Kilian, Rich Martin, and Eduardo Sontag, who provided helpful feedback on some of the earlier results presented in the dissertation.
- My colleagues directly involved in dissertation research: Emma Brunskill, Carlos Diuk, John Langford, Bethany Leffler, Michael Littman, Chris Mansley, Ali Nouri, Nick Roy, Alex Strehl, Tom Walsh, and Eric Wiewiora. In particular, I would like

to thank Michael, Alex, and Tom for their significant contributions in developing the PAC-MDP framework, the KWIK model, and related algorithms and analyses.

- My other reinforcement-learning or machine-learning research collaborators: John Asmuth, Suhrid Balakrishnan, Christopher Painter-Wakefield, Ron Parr, Gavin Taylor, Yevgeniy Vorobeychik, Jason Williams, David Wingate, Jennifer Wortman, and Tong Zhang. Although our collaborated work is not described in detail in the dissertation, working with these brilliant brains has helped me get a better big picture of the fields of reinforcement learning and machine learning.
- The RL^3 lab members, who create an inspiring, cooperative, and friendly environment to work. I will miss all the wonderful time we spent together and the many mid-night deadlines we worked so hard to meet.
- Csaba Szepesvári, who has been a consistent source of very helpful feedback on my work. I really enjoyed our conversations at conferences like ICML and NIPS.
- Proofreaders of the dissertation. I want to give warm thanks to Michael Littman and George Konidaris, who went through the whole document carefully, and to Tom Walsh and Emma Brunskill, who proofread parts of it. Their comments were tremendously helpful. Of course, I shall be responsible for all remaining mistakes and typos.
- My family members. My parents and brother have always been perfect, giving every support they could to help me establish and pursue my academic interests. They gave up *hundreds* of weekends when I was a teenager so that I could attend Olympic schools to get additional training in maths and science. I hope they will be pleased to hear that what I learned at that time has been amazingly helpful throughout the years, and particularly in various parts of my dissertation. My uncle, a distinguished mechanical engineer, inspired me with his passion for science and technologies in my childhood that eventually sent me to the graduate school. Finally, I thank my wife, Helian, who gives me a happy life full of laughs and love. Her talents and interests lie in arts, but her enthusiasm about science makes research a more enjoyable endeavor to me. I hope she enjoys my mini-lectures on computer science and maths, just as I enjoy hers on arts and culture.

Dedication

This dissertation is dedicated to my father, mother, brother, and wife. Their love, understanding, and support is priceless.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

This chapter gives an informal introduction to reinforcement learning and the class of problems we focus on in this dissertation, explains the challenges we try to solve, and then states the thesis of this dissertation, followed by a summary of contributions.

1.1 Reinforcement Learning: Achieving Intelligence by Interaction

Reinforcement learning or *RL* [Bertsekas and Tsitsiklis 1996; Kaelbling et al. 1996; Sutton and Barto 1998] is a sub-area of *artificial intelligence* [Russell and Norvig 2002], which considers how an autonomous agent situated in a possibly unknown environment may come to act intelligently by interacting/experimenting with the environment. Here, the *agent* refers to a software or hardware entity that can perceive the *state* of the environment, and take *actions* to affect the environment's state. In return, it receives a numerical signal called *reinforcement* from the environment for every action it takes. Its goal is to maximize the total reinforcements it receives over time. The so-called "reward hypothesis" says that the problem of achieving a goal can always be formulated as one of maximizing the expected value of the cumulative, real-valued reinforcement signals [Sutton 2004].

Reinforcement learning is "in a sense the whole Al problem in a microcosm" [Sutton 1992],[1] and has broad applicability. In addition to numerous natural applications in robotics (*e.g.*, Bresina et al. [2002], Tedrake et al. [2004], Ng et al. [2004], Peters and Schaal [2007]), reinforcement learning has also been be applied successfully to

[1] Reinforcement learning also arguably subsumes many extensively studied machine-learning problems [Langford and Singh 2006].

many other learning-to-control problems in a broad sense, including the early checkers-learning program [Samuel 1959], the celebrated TD-Gammon program [Tesauro 1995] that achieved world-champion-level game-playing ability in backgammon, packet routing [Boyan and Littman 1994], elevator dispatching [Crites and Barto 1996], discrete optimization problems such as job-shop scheduling and Boolean satisfiability [Zhang and Dietterich 1995; Boyan and Moore 2000; Lagoudakis and Littman 2001], channel allocation [Singh and Bertsekas 1997], algorithm selection [Lagoudakis and Littman 2000], option pricing [Tsitsiklis and Van Roy 2001], spoken dialog management [Williams 2006], resource allocation [Tesauro 2005], the game of Go [Silver et al. 2007], and parameter selection [Ruvolo et al. 2009], among numerous others.

Below, I describe some concrete examples and discuss how to formulate them as reinforcement-learning problems. We start with a simple problem that will serve as a running example in this chapter, and then move on to realistic ones.

Example 1 *Consider a minimum-cost navigation task in the directed graph in Figure 1.1. There is a set of nodes connected by edges, with node S on the left as the source and node T on the right as the sink. Each edge in the graph is associated with a real-valued cost, as labelled in the figure. In each episode, the agent starts from the source and moves along some path to the sink. In some nodes such as S and G, the agent has to take an action, that is, to decide which edge to follow. Each time it crosses an edge, the agent receives its cost. Once the sink is reached, the next episode begins. We may view reinforcements as negative costs in this example, and thus* maximizing *the total reinforcements received in an episode is equivalent to* minimizing *the total cost. In this case, the minimum-cost path from S to T is: $S \to D \to E \to F \to G \to H \to T$, and the total cost is* 3.

The simple problem above can in fact model many challenging problems including numerous examples extensively studied in the intelligent planning literature [Russell and Norvig 2002]. Furthermore, more realistic problems can be modelled if we introduce stochastic transitions to the minimum-cost navigation problem; that is, following an edge may land the agent to more than one node in a probabilistic manner. Such

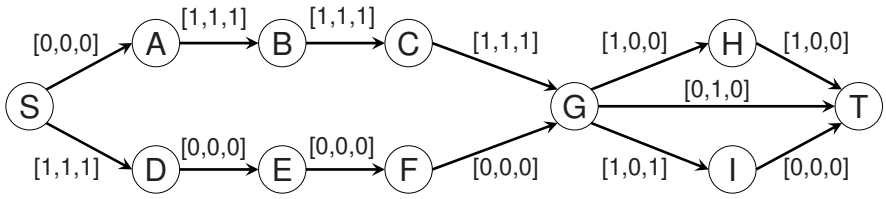


Figure 1.1: A directed graph with edge costs for minimum-cost navigation: S is the source node and T the sink.

problems are known as *stochastic shortest-path problems* that find numerous applications [Bertsekas and Tsitsiklis 1996].

Example 2 *In board games like backgammon and Go, two agents make alternating moves. The state of the game is the entire board configuration; the number of states is huge but still finite. We may assign a* 0 *reinforcement to every move an agent makes,* 1 *if an agent wins, and* -1 *if it loses. By maximizing the total reinforcements it receives, each agent effectively maximizes the chances it wins a game. After being formulated into a reinforcement-learning problem, both games can be solved by standard RL algorithms [Tesauro 1995; Silver et al. 2007]. Note that chess is a deterministic game, while backgammon is stochastic because the next board configuration depends on the result of rolling a die.*

Example 3 *In a spoken dialog system (SDS), an agent (a computer software/hardware system) participates in a spoken conversation with a person.* Dialer*—a voice dialer application accessible within the AT&T Research Labs, for example, receives daily calls from human users, asks questions (such as* ConfirmName*) to collect information about the callees, and then directs the call to the desired persons [Williams 2008].* Dialer*'s vocabulary consists of about* $50,000$ *AT&T employees. Since many employees have the same name,* Dialer *can disambiguate by asking for the callee's location. It can also disambiguate between multiple phone listings for the same person (office vs. mobile) and indicate when a callee has no number listed. In designing the system, we may assign a* -1 *reinforcement for every communicative action* Dialer *takes, a large final reward of* $+20$ *if the call is correctly transferred and a large penalty of* -20 *otherwise. By maximizing the total reinforcements* Dialer *receives in a course of conversation, it*

fulfills our design objective: it should transfer a call to the desired callee by asking as few questions as possible. After formulating the optimization of DIALER *as an RL problem, we may apply various RL algorithms to solve it [Williams 2008; Li et al. 2009c].*

In this dissertation, we focus on *risk-neutral*, *single-agent*, *online* reinforcement learning in *Markovian* environments that are *fully observable* and *stationary*. Formal definitions are given in the next chapter, but it is helpful to give informal explanations for our scope of interest.

- We focus on risk-neutral RL agents who goal is to maximize *expected* total reinforcements. Therefore, two distributions of total reinforcements are effectively identical as long as they have the same expectation. In contrast, a *risk-aversive* or *risk-sensitive* RL agent has to strike a balance between getting high reinforcements and avoiding catastrophic situations even if they happen with tiny probability (see, *e.g.*, Mihatsch and Neuneier [2002]). For example, a profit-maximizing firm may want to be conservative in making business decisions to avoid bankruptcy even if its conservation will probably lower the expected profits.
- We consider RL problems that involve decision making of a single agent so that the behavior of the environment is fixed beforehand and does not change according to what the agent does. In contrast, numerous works in multi-agent systems study interactions among multiple agents such as collaboration and competition (see, *e.g.*, Littman [1996]). Board games in Example 2 are instances of two-agent systems, while spoken dialog management in Example 3 is a natural fit of single-agent RL.
- We consider online RL problems in the sense that the interaction between the agent and environment is continuous without interruption. Consequently, the agent only perform actions in states it actually visits. In contrast, some authors have studied mechanisms such as reset (which sends the agent to a certain state) or generative model (which generates sample interactions in a state even if it is not the actual state of the environment). The online RL problem is arguably more challenging and general. More discussions are in §2.1.
- We consider Markovian environments where the *state* contains sufficient statistics

to summarize the history of the environment. In other words, the future behavior of the environment depends only on its current state and the actions taken by the agent, and is independent of its history given the current state.

- A related assumption is that the state of the environment is fully observable by the agent and thus the agent has full access to the sufficient statistics of the environment's history. In contrast, an agent in a partially observable environment (*e.g.*, partially observable Markov decision processes [Kaelbling et al. 1998]) has the additional challenge of inferring the state of the environment. In poker, for example, an agent does not observe the cards of the others, while in board games like Go the entire state is completely visible to both players.
- We only consider stationary environments whose behavior do not change over time. For instance, we require that the probability of outcomes of a coin flip does not depend on when the coin is flipped. Sometimes a non-stationary problem can be turned into a stationary one by including time as a component in the state of the environment.

All these assumptions above are reasonable approximations of many real-life problems and are common in the majority of RL literature. They simplify the problem setting and allow one to focus on many key issues in reinforcement learning that we will discuss in the next section.

1.2 Three Challenges of Reinforcement Learning

Three challenges, among others, are critical in reinforcement learning: the need for *sequential decision making*, the *exploration/exploitation dilemma*, and the need for *generalization across states*.

1.2.1 Sequential Decision Making

In reinforcement learning, an agent has to achieve its goal (that is, to maximize the total reinforcements) by taking a sequence of actions. Each action affects not only the current reinforcement the agent receives, but also the new state of the environment. Therefore,

the agent needs to maintain a balance between maximizing current reinforcement and reaching a "good" new state that allows more reinforcements to be obtained in the future. An agent that simply maximizes current reinforcement is suboptimal in almost all RL problem of practical interests.

Example 4 *Let us reconsider Example 1. In order for the agent to find a minimum-cost path from the source S to sink T in Figure 1.1, it is insufficient to behave greedily in each node by selecting the adjacent edge with the smallest cost. In node S, for instance, if the agent chooses to go to A because the immediate cost is 0 as opposed to the cost of 3 to go to D, then it has to suffer more costs by travelling from A to G. Intuitively, the long-term benefit of going to the better node D outweighs the myopic advantage of going to A.*

1.2.2 The Exploration/Exploitation Dilemma

The second challenge to designing reinforcement-learning algorithms is the *exploration/exploitation dilemma*, which is sometimes called the problem of *dual control* [Fel'dbaum 1961]. How can the agent maximize its total reinforcement if it has incomplete knowledge about the environment? Without external help, the agent needs to interact/experiment with the environment to acquire such knowledge. While the agent strives to maximize its total reinforcements, it has to purposely try actions—even if they appear suboptimal—in the hope of getting more total reinforcements in the future by obtaining more information about the environment. A purely exploring agent that extensively explores the environment is undesirable as such a utility-blind agent may suffer small reinforcements. A purely exploiting agent which always picks actions that appear the best is also undesirable as it may end up with a suboptimal action-choosing strategy because of its incomplete knowledge of the environment. Therefore, a good tradeoff between exploration and exploitation is critical.

Example 5 *In Example 1, suppose the agent knows the costs of edges along the following path: $S \to D \to E \to F \to G \to T$, but not the costs of others. It has to explore an edge by actually crossing it. If the agent is afraid of potential cost increase*

incurred by exploration, it may content itself with the path above and stick to it in every episode. By doing so, the agent avoids taking the worse path to T via I. However, such a conservative agent will miss the actual optimal path with insufficient exploration, and will suffer more costs in the long run.

The need for exploration also exists in other machine-learning settings such as *active learning*, in which a supervised-learning agent decides what input–output data to use in its training set; see Settles [2009] for a recent survey. As we will see in §4, the exploration/exploitation tradeoff is significantly harder in sequential decision making. On the other hand, techniques developed for exploration in RL may find use in other machine-learning problems as well.

1.2.3 Generalization

The third challenge of reinforcement learning, which is also faced by other machine-learning and artificial-intelligence problems, is how to generalize across states. The game of backgammon, for example, has a gigantic state space (in which each state is a board configuration), rendering reinforcement learning hopeless unless approximation techniques are used to enable generalization [Tesauro 1995].

Consider the following example for concreteness.

Example 6 *To find a minimum-cost path in problems like Figure 1.1 when edge costs are unknown, the agent has to cross all edges at least once in the worst case, rending this approach inefficient in graphs with many edges. However, if the agent has access to additional information to generalize its observation of costs of some edges to others, exploration may be done more efficiently. Figure 1.2 depicts a directed graph, in which each edge is associated with a* cost vector *of dimension $n = 3$. The cost of traversing an edge is the dot product of its cost vector with a fixed weight vector $\mathbf{w}^* = [1, 2, 0]^\top$. The resulting edge costs coincide with those in Figure 1.1.*

Now suppose $\mathbf{w}^*$ is unknown to the agent, but the graph topology and all cost vectors are. Also, we suppose the agent has explored the following edges and thus observed the corresponding costs: $(G, H), (G, I), (G, T)$. These "training data" allow the agent to

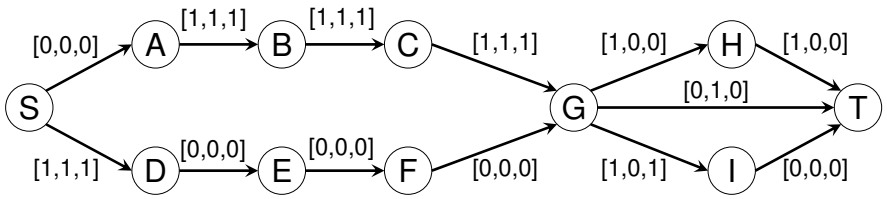


Figure 1.2: A directed graph with edge-cost vectors for minimum-cost navigation: S is the source node and T the sink. The cost of an edge is the dot product between its cost vector and the vector $\mathbf{w}^* = [0, 1, 2]^\top$. The resulting edge costs are identical to those in Figure 1.1.

solve the following system of linear equations

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{w}^* = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}$$

for $\mathbf{w}^*$ *directly. Thus, the agent is able to infer all edge costs—without actually crossing all edges—and then find the minimum-cost path. This example shows how generalization of knowledge from one state/node to others may result in faster learning.*

While function approximation such as linear regression, neural networks, and decision trees has been extensively studied in supervised learning (see, *e.g.*, Hastie et al. [2003]), it is much harder to be applied in reinforcement learning, partly due to the sequential decision making nature of reinforcement-learning problems and the need for balancing exploration and exploitation.

1.3 Thesis

The challenges discussed in the previous section suggest the difficulty of the central question studied in this dissertation: Can we devise reinforcement-learning algorithms that *provably efficiently* solve the exploration/exploitation dilemma when function approximation may be used?

The dissertation investigates a computational learning model called *Knows What It Knows* or *KWIK* that is suitable for use in reinforcement-learning algorithms. A key characteristic of a KWIK learner is the option of explicitly saying "I don't know"

when the learner is unable to make a sufficiently accurate prediction. Naturally, this knows-what-it-knows property is useful for an RL agent to decide what to explore to reduce uncertainty about the environment, as illustrated in the following example.

Example 7 *In Example 6, the agent is able to infer* $\mathbf{w}^*$ *if it has crossed the edges* $(G, H), (G, I), (G, T)$*. However, we did not address the question how the agent decide which edge to explore and which not to. Again, suppose* $\mathbf{w}^* \in \mathbb{R}^3_+$ *is unknown to the agent, but the graph topology and all cost vectors are.*

First, the agent can deduce that the path, $S \to A \to B \to C \to G$*, is more costly than the alternative,* $S \to D \to E \to F \to G$*, so the agent can safely choose the latter to arrive at* G*. From node* G*, there are three distinct paths to reach* T*:* $G \to H \to T$*,* $G \to T$*, and* $G \to I \to T$*. Now, the agent has to decide which edge to follow.*

A simple approach for this task is for the agent to assume edge costs are uniform and walk the shortest (middle) path to collect the following data:

$$\mathbf{w}^* \cdot [1, 1, 1] = 3, \qquad \mathbf{w}^* \cdot [0, 1, 0] = 1.$$

Standard linear least-squares regression could use this dataset to find the following solution: $\hat{\mathbf{w}} = [1, 1, 1]^\top$*. The learned weight vector could then be used to estimate costs for the three paths from* G *to* T*:* 2 *for the top,* 1 *for the middle, and* 2 *for the bottom. Using these estimates, an agent would continue to take the middle path forever, never realizing it is not optimal.*

In contrast, consider a learning algorithm that "knows what it knows". Instead of creating an approximate weight vector $\hat{\mathbf{w}}$*, it reasons about whether the costs for each edge can be obtained from the available data. The middle path, since all its edge costs have been observed, is definitely* 1*. The last edge of the bottom path has cost vector* $[0, 0, 0]$*, so its cost must be* 0*, but the penultimate edge of this path has cost vector* $[1, 0, 1]$*. This vector is a linear combination of the two observed cost vectors, so, regardless of* $\mathbf{w}^*$*, its cost is*

$$\mathbf{w}^* \cdot [1, 0, 1] = \mathbf{w}^* \cdot ([1, 1, 1] - [0, 1, 0]) = \mathbf{w}^* \cdot [1, 1, 1] - \mathbf{w}^* \cdot [0, 1, 0] = 3 - 1 = 2.$$

Thus, the agent knows the bottom path's cost is 2*—worse than the middle path.*

The vector $[1, 0, 0]$ *on the top path is linearly independent of the observed cost vectors, so its cost is undecided. We know we do not know. A safe thing to assume provisionally is that the cost is zero—the smallest possible cost, encouraging the agent to try the top path in the next episode. Now, it observes* $\mathbf{w}^* \cdot [1, 0, 0] = 0$*, allowing it to infer* $\mathbf{w}^*$ *exactly and accurately predict the cost for any vector (since the training data spans* $\mathbb{R}^3$*). It now knows that it knows all the costs, and can confidently take the optimal (top) path.*

In general, any algorithm that guesses a weight vector may never find the optimal path. An algorithm that uses linear algebra to distinguish known from unknown costs will either take an optimal route or *discover the cost of a linearly independent cost vector on each episode. Thus, it can never choose suboptimal paths more than* n *times. Formal discussions of learning noise-free linear functions are provided in §5.2.3.*

In contrast, an agent that does not generalize, but visits every edge to learn its cost, will require m *episodes to learn optimal behavior, in the worst case, where* m *is the number of edges in the graph. This example shows how combining generalization with explicitly distinguished known and unknown areas can lead to efficient and optimal decision algorithms.*

As indicated in the example above, KWIK provides a useful mechanism for efficient exploration. We argue that it does capture the necessary ingredients for efficient reinforcement learning, and indeed relate KWIK learning to efficient reinforcement learning in a general way. We study some of the basic properties of KWIK learning, and then use it to unify many existing provably efficient reinforcement-learning algorithms as well as to propose new ones. My thesis is the following:

> *The KWIK learning model provides a flexible, modularized, and unifying way for creating and analyzing reinforcement-learning algorithms with provably efficient exploration.*

1.4 Contributions

The rest of the document is divided into three parts, followed by a concluding chapter and two appendices: Part I (§§2–4) reviews background of planning and learning

in Markov decision processes; Part II (§§5–6) introduces the KWIK learning model; Part III (§§7–8) uses tools from the KWIK model to create, analyze, and unify many provably efficient reinforcement-learning algorithms. Each chapter and the major contributions are outlined as follows.

- §2 reviews the basic theory of Markov decision processes (MDPs)—the class of environments we focus on, and then defines the "planning" and "learning" problems in an MDP. Most results in this chapter are found in standard textbooks on this topic, such as the ones by Puterman [1994] and by Bertsekas [2001].

- §3 surveys planning algorithms in MDPs, including *exact* algorithms for finite MDPs and *approximate* algorithms that are more general and suitable for general MDPs. New results in this chapter include:

  - ◇ A convergence proof of the prioritized sweeping algorithm and some of its variants [Li and Littman 2008b].
  - ◇ A unified notation and theory of state abstraction for MDPs [Li, Walsh, and Littman 2006].

- §4 surveys reinforcement learning in MDPs. First, classic examples of exact and approximate RL algorithms are reviewed. Then, the exploration/exploitation dilemma is introduced, which motivates the notion of PAC-MDP as a mathematical framework for formal analysis of exploration efficiency of RL algorithms. New results in this chapter include:

  - ◇ The notion of PAC-MDP for formal analysis of RL algorithms, and a general PAC-MDP theorem that generalizes the original one studied by Strehl, Li, and Littman [2006a]. Furthermore, the theorem improves its ancestor via a more careful analysis, yielding a better bound that will be used in §§7–8 to improve a number of existing PAC-MDP results.
  - ◇ An example showing that Bayesian exploration may not be PAC-MDP although it is optimal in its own sense.

- §5 introduces the KWIK learning model, presents a few KWIK-learnable examples, and studies its basic properties. New results in this chapter include:
  - ◇ A formal definition of the KWIK model [Li, Littman, and Walsh 2008].
  - ◇ A number of KWIK-learnable examples, both deterministic and stochastic, together with complete algorithmic details and analyses. Some of them improve or extend previous results by Li, Littman, and Walsh [2008] and Brunskill, Leffler, Li, Littman, and Roy [2008].
  - ◇ A preliminary discussion of basic properties of the KWIK framework.
- §6 studies general techniques for constructing algorithms for complex KWIK problems by combining base KWIK algorithms. New results in this chapter include:
  - ◇ Full details and improved analyses of a few general KWIK combination techniques [Li, Littman, and Walsh 2008] including input-partition, output-combination, cross-product, union, and noisy union.
  - ◇ A matching lower bound showing the optimality of noisy union in terms of sample complexity [Diuk, Li, and Leffler 2009].
  - ◇ An algorithm for KWIK-learning multivariate normal distributions with unknown mean vectors and covariance matrices, generalizing previous results by Brunskill, Leffler, Li, Littman, and Roy [2008] and others.
- §7 studies and unifies model-based PAC-MDP reinforcement learning, in which the agent estimates a model of the environment from observations and then uses the model to choose actions. New results in this chapter include:
  - ◇ A novel, abstract algorithm, KWIK-Rmax, which is shown to be PAC-MDP whenever the transition and reward functions of the underlying MDP can be KWIK-learned.
  - ◇ A unification of almost all existing model-based PAC-MDP algorithms, based on KWIK-Rmax, including those for finite MDPs, MDPs with linear dynamics, MDPs with normal offset dynamics, and factored-state MDPs modeled as a dynamical Bayes net (DBN).

  - ◇ A new PAC-MDP algorithm, met-Rmax, which can learn the structure of the DBN while efficiently exploring a factored-state MDP, improves on a state-of-the-art algorithm both analytically and empirically [Diuk, Li, and Leffler 2009].

- §8 studies model-free PAC-MDP reinforcement learning, in which the agent decides what action to take without estimating a model of the environment. New results in this chapter include:
  - ◇ A general lemma relating Bellman error and a condition for PAC-MDP for algorithms that acts greedily with respect to its value function. The lemma is used in the PAC-MDP analysis of three algorithms, which are simplified versions of their ancestors [Strehl, Li, and Littman 2006b; Strehl, Li, Wiewiora, Langford, and Littman 2006c].
  - ◇ The first model-free PAC-MDP algorithm, delayed Q-learning, for finite MDPs. The algorithm simplifies the original one of Strehl, Li, Wiewiora, Langford, and Littman [2006c].
  - ◇ A new lower bound of sample complexity of exploration for RL algorithms, which matches the sample complexity upper bound of delayed Q-learning in terms of several factors, including the probably most important one (number of states), and thus showing the optimality of delayed Q-learning in this sense.
  - ◇ A new algorithm, LSPI-Rmax, which combines the algorithmic insights of Rmax with the model-free least-squares policy iteration algorithm [Li, Littman, and Mansley 2009a]. The algorithm has demonstrated promising empirical results in a few benchmark problems, suggesting PAC-MDP algorithms and analysis are useful for creating practical RL algorithms that work well empirically.
  - ◇ A new algorithm, REKWIRE, which is model-free and uses linear function approximation. We show REKWIRE is PAC-MDP under certain assumptions about the linear approximation architecture [Li and Littman 2008a].

- §9 concludes the dissertation, including implications concerning the strengths and weaknesses of model-based and model-free RL algorithms. A number of open problems, extensions, and limitations of our work are discussed.

- Supporting materials are provided in the appendices. §A lists the notation and convention we adopt in the dissertation. §B collects a number of mathematical facts that are used in our analysis.

Part I

Planning and Learning in Markov Decision Processes

Chapter 2

Markov Decision Processes

In reinforcement learning, an environment is often modeled as a *Markov decision process* or *MDP* [Puterman 1994], where the history of the environment can be summarized in a sufficient statistic called *state*. We first introduce Markov decision processes, define (optimal) policies and value functions, and then give formal definitions of the planning and learning problems in an MDP.

2.1 Definition

In reinforcement learning, an environment is often modeled as a *Markov decision process* or *MDP* [Puterman 1994], where the history of the environment can be summarized in a sufficient statistic called *state*. MDPs are natural abstraction of many real-life problems such as those investigated by Puterman [1994].

A Markov decision process is defined as a five-tuple: $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, where:

- $\mathcal{S}$ is the *state space*. It can be discrete or continuous.
- $\mathcal{A}$ is the *action space*. Similarly, it can be discrete or continuous.
- $T \in (\mathcal{P}_{\mathcal{S}})^{\mathcal{S}\times\mathcal{A}}$ is a *transition function*, with $\mathcal{P}_{\mathcal{S}}$ denoting the set of probability distributions over $\mathcal{S}$. If $\mathcal{S}$ is discrete, we may define $T(\cdot \mid s, a)$ for any $(s, a) \in \mathcal{S} \times \mathcal{A}$ as a probability *mass* function, so that $T(s' \mid s, a)$ is understood to be the probability of reaching a new state s' if action a is executed in state s. If $\mathcal{S}$ is continuous, $T(\cdot \mid s, a)$ is understood to be a probability *density* function.
- $R \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ is a *reward function* that defines the reinforcements received by the agent. For now on, we will use the word "reward" instead of "reinforcement" as the former is used more often in the literature. In most practical situations, we may assume, without loss of generality, that R is bounded: $R \in [0, 1]^{\mathcal{S}\times\mathcal{A}}$;

otherwise, policy-invariant reward transformation techniques [Ng et al. 1999] may be used to transform the reward to the range $[0, 1]$ when *a priori* bounds are known for the reward function.

- $\gamma \in (0, 1)$ is a *discount factor*, whose role will be discussed shortly.

In the minimum-cost navigation task of Example 1, states are the nodes in the graph, actions are the edges, and rewards are the negative costs. In the spoken dialog management problem of Example 3, a state contains the agent's belief distribution of the user intention and auxiliary information regarding the status of the conversation, an action may be a communicative action (like asking whom the caller is trying to reach), and the reward function is as given before. However, it is not appropriate to model multi-agent problems like those in Example 2 as an MDP.

Definition 1 *The* online interaction *between the agent and environment, modeled as an MDP* $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$*, proceeds according to the following protocol. Beginning at the initial timestep* $t = 1$*, an online interaction between the agent and the environment proceeds in* discrete timesteps *as follows. At timestep* $t = 1, 2, 3, \ldots$*,*

1. *The agent perceives the* current state $s_t \in \mathcal{S}$ *of the environment, and takes an action* $a_t \in \mathcal{A}$*.*
2. *In response, the environment sends an immediate reward* $r_t \in [0, 1]$ *to the agent, and moves to a* next state $s_{t+1} \in \mathcal{S}$*. This* transition *is governed by the dynamics of the MDP. In particular, the expectation of* r_t *is* $R(s_t, a_t)$*, and the next state* s_{t+1} *is drawn randomly from the distribution* $T(\cdot \mid s_t, a_t)$*.*
3. *The clock ticks:* $t \leftarrow t + 1$*.*

Our discussion will be easier with the following terminology:

- An MDP is called *finite* if both the state and action spaces are finite sets; in contrast, it is called *continuous* if either the state or action space is continuous.
- An MDP is often referred to as an *uncontrolled MDP* or a *Markov chain* if $|\mathcal{A}| = 1$. For a Markov chain, we may drop the dependence on a denote the transition function and reward function by $T(\cdot \mid s)$ and $R(s)$, respectively, since there is only one action.

- An MDP is *stochastic* if the reward and transitions are randomized; otherwise, it is *deterministic*. For a deterministic MDP, we abuse notation by using $T(s, a)$ to denote the next state of the agent if it takes action a in state s.

2.2 Policy and Value Function

The return of the agent at timestep t, denoted R_t, is defined as the total discounted reward received by the agent after time t:

$$R_t \stackrel{\text{def}}{=} \sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau}.$$

Mathematically, the discount factor is a convenient trick that guarantees boundedness of the return provided that all immediate rewards are bounded. Practically, it says that a unit reward in the next timestep is worth γ in the current timestep, which has useful interpretations in many real-life problems [Puterman 1994].

The return R_t is defined using an infinite sum, which may result in difficulty in analysis. The following useful lemma states that it can be approximated to arbitrary precision by a partial sum of a "small" number of leading terms.

Lemma 1 *Let $R_t(H)$ be a H-step discounted return at timestep t defined by*

$$R_t(H) \stackrel{\text{def}}{=} \sum_{\tau=0}^{H-1} \gamma^\tau r_{t+\tau}.$$

Then for any $\epsilon > 0$ and t, we have $0 \le R_t - R_t(H) \le \epsilon$ when

$$H \quad \ge \quad \frac{1}{1-\gamma} \ln \frac{1}{\epsilon(1-\gamma)}. \tag{2.1}$$

Proof. It is clear that $R_t \ge R_t(H)$ as all immediate rewards are non-negative. On the other hand, if H satisfies Equation 2.1, then the inequality $\gamma \ge 1 + \ln \gamma$ implies

$$H \ge \frac{\ln \frac{1}{\epsilon(1-\gamma)}}{\ln \frac{1}{\gamma}} = \log_\gamma \left(\epsilon(1-\gamma)\right),$$

and hence

$$R_t - R_t(H) = \sum_{\tau=H}^{\infty} \gamma^\tau r_{t+\tau} \le \sum_{\tau=H}^{\infty} \gamma^\tau = \frac{\gamma^H}{1-\gamma} \le \frac{\epsilon(1-\gamma)}{1-\gamma} = \epsilon.$$

as all immediate rewards are bounded by 1 from above. □

Clearly, the return R_t is a random variable that depends on a few factors: the transition and reward functions of the MDP as well as the agent's action-selection rule. The first two factors are inherent in the environment and cannot be altered by the agent. However, by varying the way it chooses actions in the interaction, the agent is able to control or maximize the return. Formally, a *stationary policy*, $\pi \in \mathcal{A}^{\mathcal{S}}$, defines an action-selection rule, where $\pi(s)$ is the action to be chosen when the current state is s. Policies are also called *feedback laws* or *control laws* in the literature.

For any MDP M, a stationary policy π determines a distribution of the reward sequence, and thus a distribution of the return. Therefore, we may talk about expected return and use it to evaluate a policy. Naturally, a reward-maximizing agent prefers policies that yield largest expected returns in all states. We define the *state-value function*, $V_M^\pi(s)$, as the expected return by executing π starting from state s:

$$V_M^\pi(s) \quad \stackrel{\text{def}}{=} \quad \mathbf{E}_\pi \left[R_1 \mid s_1 = s \right], \tag{2.2}$$

where $\mathbf{E}_\pi$ refers to the probability distribution of the reward sequence induced by the dynamics of the MDP as well as the policy (*i.e.*, $a_t = \pi(s_t)$ for all t). Similarly, the *state–action value function*, $Q_M^\pi(s, a)$, is the expected return by taking action a in state s and following π thereafter:

$$Q_M^\pi(s, a) \quad \stackrel{\text{def}}{=} \quad \mathbf{E}_\pi \left[R_1 \mid s_1 = s, a_1 = a \right], \tag{2.3}$$

To maximize the total rewards received from the environment, the agent desires an *optimal policy* π^* whose value functions, denoted by $V_M^*(s)$ and $Q_M^*(s, a)$, respectively, satisfy the conditions:

$$\begin{aligned} V_M^*(s) &= \max_\pi V_M^\pi(s) \\ Q_M^*(s) &= \max_\pi Q_M^\pi(s). \end{aligned}$$

The (optimal) state–action value functions are frequently referred to as *(optimal) Q-functions* in the literature. If there is no ambiguity, we drop the subscript M from the value functions and simplify the notation to V^π, Q^π, V^*, and Q^*, respectively.

It is clear that the value functions defined above must be bounded between 0 and $1/(1-\gamma)$, since any return satisfies

$$R_1 = \sum_{t=1}^{\infty} \gamma^{t-1} r_t \leq \sum_{t=1}^{\infty} \gamma^{t-1} = \frac{1}{1-\gamma}.$$

In some applications, however, expert knowledge is available about a tighter upper bound of the value function of an MDP. We denote this upper bound by $V_{\max}$, and will use it in the analysis in later chapters to yield tighter results. It should be understood that, if no such expert knowledge is available, we may simply set $V_{\max}$ to $1/(1-\gamma)$.

As an analogue of Lemma 1, we may define H-step value functions in a straightforward manner, which are denoted $V^\pi(s, H)$ and $Q^\pi(s, a, H)$, respectively. Lemma 1 implies the following result immediately:

Lemma 2 *For any $\epsilon > 0$, $s \in \mathcal{S}$, $a \in \mathcal{A}$, and policy π, we have $0 \leq V^\pi(s) - V^\pi(s, H) \leq \epsilon$ and $0 \leq Q^\pi(s, a) - Q^\pi(s, a, H) \leq \epsilon$, when*

$$H \geq \frac{1}{1-\gamma} \ln \frac{1}{\epsilon(1-\gamma)}.$$

The existence of (optimal) value functions and policies is a well-studied problem. They always exist for finite MDPs, and also exist under certain technical assumptions for MDPs with infinite (either discrete or continuous) state or action spaces [Bertsekas and Shreve 1996; Puterman 1994]. It is beyond the scope of the thesis to identify sufficient conditions for their existence, and to simplify exposition, we will always assume such quantities exist whenever they are used.

A greedy policy π_V (or π_Q) with respect to a value function V (or Q) is defined by

$$\pi_V(s) \stackrel{\text{def}}{=} \operatorname*{argmax}_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s \in \mathcal{S}} T(s' \mid s, a) V(s') \right) \tag{2.4}$$

$$\pi_Q(s) \stackrel{\text{def}}{=} \operatorname*{argmax}_{a \in \mathcal{A}} Q(s, a). \tag{2.5}$$

An important fact is that an optimal policy must be greedy with respect to the optimal value functions:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s \in \mathcal{S}} T(s' \mid s, a) V^*(s') \right) \tag{2.6}$$

$$= \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s, a). \tag{2.7}$$

It is often easier to work with the state–action value function than state-value functions. Most of the results presented in this dissertation are therefore stated in terms of state–action value functions.

The following lemma, which is proved by Singh and Yee [1994], states that the greedy policy with respect to a sufficiently accurate value function is uniformly near-optimal.[1]

Lemma 3 *Let $Q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ be a state–action value function, and π_Q the greedy policy with respect to Q. Then,*

$$V^*(s) - V^{\pi_Q}(s) \leq \frac{\|Q - Q^*\|_\infty}{1-\gamma}.$$

The policies we have discussed so far are *deterministic* and *stationary* policies. Two other kinds of policies will be useful for our purpose, which we will define below. Value functions as in Equations 2.2 and 2.3 can be defined in the same way, although it is known that there exists at least one optimal policy that is both deterministic and stationary [Puterman 1994].

A *stochastic* and stationary policy is one that selects actions randomly. Specifically, it maps states to probability distributions over the set of actions; $\pi \in (\mathcal{P}_\mathcal{A})^\mathcal{S}$. We will use $\pi(a \mid s)$ to denote the probability of choosing a in state s when π is stochastic.

Another type of policy is those that are *non-stationary* in the sense that they do not map states to actions (or probability distributions over actions, in the case of a stochastic policy). Rather, it maps a history of interaction to actions. Formally, a deterministic, non-stationary policy at timestep t is: $\pi_t \in \mathcal{A}^{(\mathcal{S}\times\mathcal{A}\times\mathbb{R})^{t-1}\times\mathcal{S}}$, where the first $(t-1)$ factors of $\mathcal{S}\times\mathcal{A}\times\mathbb{R}$ refers to the visited states, chosen actions, and observed rewards in timesteps $1, 2, \ldots, t-1$, and the last $\mathcal{S}$ corresponds to the current state.

Finally, we mention a useful observation that, when the agent follows a stationary policy π, then the sequence of states of state–actions may be viewed as states of an induced Markov chain. For instance, if π is stochastic, then the induced Markov chain M' has an expanded state space, $\mathcal{S}' \stackrel{\text{def}}{=} \mathcal{S}\times\mathcal{A}$, and the transition and reward functions,

[1] Singh and Yee [1994] considers finite MDPs only, but their proof may be adapted to general MDPs.

T' and R', are defined respectively by:

$$\begin{aligned} T'\left((s',a') \mid (s,a)\right) &\stackrel{\text{def}}{=} T(s' \mid s,a)\pi(a' \mid s') \\ R'\left((s,a)\right) &\stackrel{\text{def}}{=} R(s,a). \end{aligned}$$

2.3 Bellman Equations

Bellman equations characterize the structure of optimal value functions and embody the *principle of dynamic programming* [Bellman 1957].

First, we define the *Bellman operator* (*a.k.a. Bellman backup*) $\mathfrak{B} \in \left(\mathbb{R}^{\mathcal{S}}\right)^{\mathbb{R}^{\mathcal{S}}}$ by

$$\mathfrak{B}V(s) \stackrel{\text{def}}{=} \max_{a\in\mathcal{A}} \mathfrak{B}^a V(s), \quad \forall s \in \mathcal{S} \tag{2.8}$$

where the operator $\mathfrak{B}^a$ associated with action a is defined by

$$\mathfrak{B}^a V(s) \stackrel{\text{def}}{=} R(s,a) + \gamma \sum_{s'\in\mathcal{S}} T(s' \mid s,a)V(s'), \quad \forall s \in \mathcal{S}. \tag{2.9}$$

In other words, $\mathfrak{B}^a V$ defines a new value function where $\mathfrak{B}^a V(s)$ is the one-step lookahead value of state s using $V(\cdot)$ to retrieve the value of the successor states s'; $\mathfrak{B}V(s)$ is the maximum one-step lookahead value. Clearly, properties of $\mathfrak{B}$ also holds for $\mathfrak{B}^a$ since $\mathfrak{B}^a$ can be viewed as the Bellman operator in a new MDP whose action set is the singleton $\{a\}$. By abusing notation, we also use $\mathfrak{B}$ to denote the Bellman operator for action–value functions: given any action–value function $Q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$,

$$\mathfrak{B}Q(s,a) \stackrel{\text{def}}{=} R(s,a) + \gamma \sum_{s'\in\mathcal{S}} T(s' \mid s,a) \max_{a'\in\mathcal{A}} Q(s',a'). \tag{2.10}$$

It is easy to see that the Bellman operator is *monotonic* in the following sense:

Lemma 4 *Let $V_1, V_2 \in \mathbb{R}^{\mathcal{S}}$ be two bounded state-value functions and $V_1 \le V_2$, then $\mathfrak{B}V_1 \le \mathfrak{B}V_2$. Similarly, let $Q_1, Q_2 \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ be two bounded state–action value functions and $Q_1 \le Q_2$, then $\mathfrak{B}Q_1 \le \mathfrak{B}Q_2$.*

Proof. Assume $V_1 \le V_2$, and let $s \in \mathcal{S}$ be any state. Then for any $a \in \mathcal{A}$,

$$\mathfrak{B}^a V_1(s) - \mathfrak{B}^a V_2(s) = \gamma \sum_{s'\in\mathcal{S}} T(s' \mid s,a)\left(V_1(s') - V_2(s')\right) \le 0,$$

and so

$$\mathfrak{B}V_1(s) - \mathfrak{B}V_2(s) = \max_{a\in\mathcal{A}} \mathfrak{B}^a V_1(s) - \max_{a\in\mathcal{A}} \mathfrak{B}^a V_2(s) \leq \max_{a\in\mathcal{A}} (\mathfrak{B}^a V_1(s) - \mathfrak{B}^a V_2(s)) \leq 0,$$

where the inequality above makes use of the fact that $\max_x f_1(x) - \max_x f_2(x) \leq \max_x(f_1(x) - f_2(x))$. The inequality for Q can be proved similarly. □

A fundamental property of the Bellman operator is the contraction property, which is stated formally in the following lemma.

Lemma 5 *[Puterman 1994, Proposition 6.2.4] Let $V_1, V_2 \in \mathbb{R}^{\mathcal{S}}$ be two bounded state-value functions, then,*

$$\|\mathfrak{B}V_1 - \mathfrak{B}V_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty .$$

Similarly, let $Q_1, Q_2 \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ be two bounded state–action value functions, then,

$$\|\mathfrak{B}Q_1 - \mathfrak{B}Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty .$$

The contraction property guarantees existence of a *fixed point*, according to Banach's Fixed-Point Theorem [Puterman 1994, Theorem 6.2.3]. The fixed point coincides with the optimal value function V^* (or Q^*). Therefore, computing V^* (or Q^*) of a given MDP is equivalent to solving the so-called *Bellman equation*:

$$V = \mathfrak{B}V \tag{2.11}$$

$$Q = \mathfrak{B}Q. \tag{2.12}$$

If a value function does not solve the Bellman equation exactly, namely, $V \neq V^*$ (or $Q \neq Q^*$), then it incurs a nonzero *Bellman error* (or *Bellman residual*):

$$E(s;V) \stackrel{\text{def}}{=} \mathfrak{B}V(s) - V(s). \tag{2.13}$$

$$E(s,a;Q) \stackrel{\text{def}}{=} \mathfrak{B}Q(s,a) - Q(s,a). \tag{2.14}$$

An important special case of Bellman equations is when a fixed policy is used, making the MDP an induced Markov chain and the corresponding Bellman operator a linear operator. For instance, when the MDP has finitely many states and a fixed policy π is used, then the functions V^π solve the following system of linear equations:

$$V(s) = R(s,\pi(s)) + \gamma \sum_{s'\in\mathcal{S}} T(s' \mid s,\pi(s))V(s'), \tag{2.15}$$

and similarly for V^π:

$$Q(s,a) \quad = \quad R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) Q(s', \pi(s')). \tag{2.16}$$

Finally, we give a useful lemma that follows immediately from the definition of $\mathfrak{B}$.

Lemma 6 *Let $V_1, V_2 \in \mathbb{R}^{\mathcal{S}}$ be two bounded state-value functions such that $V_1(s) = V_2(s) + c$ for all $s \in \mathcal{S}$ where $c \in \mathbb{R}$ is a constant, then,*

$$\mathfrak{B}V_1(s) - \mathfrak{B}V_2(s) = \gamma c, \quad \forall s \in \mathcal{S}.$$

Similarly, let $Q_1, Q_2 \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ be two bounded state–action value functions such that $Q_1(s,a) = Q_2(s,a) + c$ for all (s,a) where $c \in \mathbb{R}$ is a constant, then,

$$\mathfrak{B}Q_1(s,a) - \mathfrak{B}Q_2(s,a) = \gamma c, \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}.$$

2.4 Planning and Learning

An implication of Equation 2.7 is that the agent can act optimally if it knows the optimal value function Q^*. In fact, most algorithms for solving MDPs (*i.e.*, finding π^*) work along this line of reasoning by trying to approximate Q^*, whose asymptotic correctness is guaranteed by Lemma 3. We make an important distinction between two problems—planning and learning—when we talk about solving an MDP.

Definition 2 *Let $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ be an MDP. The* planning problem *is one of computing an optimal policy π^* of M when the complete five-tuple is given as input to the agent.*

Definition 3 *Let $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ be an MDP. The* reinforcement-learning problem *is similar to the Planning problem except: (i) the transition and reward functions of M are not provided as input; and (ii) the agent must infer these dynamics (either explicitly or implicitly) from the transitions it experiences during interactions with the MDP.*

Various models exist to capture the ways the agent interacts with the MDP. This dissertation focuses on the most challenging one of *online interactions* (Definition 1),

in which the agent can only take actions in the state in which it is situated. I briefly mention other somewhat easier models of interactions:

- The *reset model* allows the agent to reset its state to a fixed state $s_0 \in \mathcal{S}$, which is sometimes called the start state. In this model, finding an optimal policy can be reduced to a sequence of supervised-learning problems [Fiechter 1994]. This model will be used in §8.4, but not in other parts of the dissertation.

- The *generative model* allows the agent to send a query (s, a) to an oracle $\mathfrak{O}$, which returns a sample (r, s') so that $\mathbf{E}[r] = R(s, a)$ and $s' \sim T(\cdot \mid s, a)$. This model significantly simplifies the exploration/exploitation tradeoff: when the agent chooses an unnecessary state–action pair to sample, this mistake does not influence the future states of the agent. In contrast, a mistake made by an online agent may require a large number of actions to fix; see Example 8 for an example. Since a generative model can be viewed as an implicit specification of the MDP model, reinforcement learning under the generative-model assumption is in fact very similar to sampling-based approximate planning (*c.f.*, §3.3).

- The *parallel sampling model* [Kearns and Singh 1999] is closely related to the generative model: the agent may submit a argumentless query to an oracle $\mathfrak{O}$, and receives a sample transition for *every* state–action pair:

$$\mathcal{D} = \left\{(s, a, r, s') \mid s \in \mathcal{S}, a \in \mathcal{A}, \mathbf{E}[r] = R(s, a), s' \sim T(\cdot \mid s, a)\right\}.$$

  Clearly, a parallel sampler can be simulated by a generative model; on the other hand, a parallel sampler can be used as a generative model, although all sample transitions are wasted except the one for the queried state–action pair.

- The *batch learning* model requires that the agent optimizes its policy based on a *static* set of sample transitions,

$$\mathcal{D} = \{(s_i, a_i, r_i, s'_i) \mid \mathbf{E}[r_i] = R(s_i, a_i), s'_i \sim T(\cdot \mid s_i, a_i), i = 1, 2, \ldots, m\}.$$

  Due to the limited access of the model dynamics (through the finite sample set $\mathcal{D}$), the agent may not always find a (near-)optimal policy. On the other hand,

by separating policy optimization from data collection, it is often easier to create and analyze algorithms (*e.g.*, Lagoudakis and Parr [2003a] and Schuurmans and Greenwald [1999]).

Planning and learning are closely related given their similarities [Sutton and Barto 1998, Chapter 9]. For ease of exposition, however, they are treated separately here. The next two chapters survey a number of planning and learning algorithms in MDPs.

Chapter 3

Planning in Markov Decision Processes

Planning is sometimes an important sub-step in reinforcement learning. In fact, it can be viewed as a simpler version of reinforcement learning with access to the MDP model. Planning in MDPs often involves computing or approximating the optimal value function. This chapter surveys both exact and approximate planning algorithms.

In MDP planning, we typically assume the agent has access to the dynamics of the MDP. Equations 2.6 and 2.7 imply that we may approximate either V^* or Q^* to approximate the optimal policy. Therefore, depending on convenience or convention, some algorithms compute V^* while others compute Q^*.

3.1 Exact Planning in Finite Markov Decision Processes

This section summarizes a few classic planning approaches to finite MDPs, where we can afford to compute and represent the *exact* optimal value function and policy. The focus of this section is on a class of methods called *dynamic programming* [Bellman 1957], while a linear-programming-based approach is also described.

Since there are only finitely many states and actions, we assume the value functions and policies are represented by lookup tables. Let $n = |\mathcal{S}|$ and $m = |\mathcal{A}|$ be the numbers of states and actions, respectively. If we number the states and actions so that $\mathcal{S} = \{1, 2, \ldots, n\}$ and $\mathcal{A} = \{1, 2, \ldots, m\}$, then the Q-function can be stored in a matrix with n rows and m columns, with the (s, a)-entry storing the value of $Q(s, a)$.

3.1.1 Value Iteration

Value iteration [Bellman 1957] is probably the simplest and easiest-to-implement algorithm for solving an MDP. This algorithm operates in the value-function space. Starting with an arbitrary, bounded initial value-function estimate, it repeatedly applies the Bellman operator (§2.3) so that the value-function estimate approaches the optimal value function in the limit.

The basic form of value iteration is given in Algorithm 1. For concreteness, the pseudocode initializes the value function to be zero everywhere, but value iteration is guaranteed to converge to the optimal value function for any bounded initial value function. Let V_t be the value-function estimate before the t-th iteration in Algorithm 1. A well-known fact of value iteration is that the sequence of value functions, $[V_t]_{t\in\mathbb{N}}$, approaches V^* at a geometric rate, which follows from the contraction property of the Bellman operator (Lemma 5).

It should be noted that, even if V_t converges to V^* only in the limit, in practice, we can terminate the algorithm after iteration t when we discover the change of value function, $\|V_{t+1} - V_t\|_\infty$, drops below a threshold [Williams and Baird 1993, Theorem 3.1]. In fact, it can be shown that the greedy policy π_t of V_t in value iteration will converge to π^* after finitely many iterations even if V_t may not equal V^* exactly [Puterman 1994, Theorem 6.3.3].

Algorithm 1 Value iteration

0: **Inputs:** $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
1: Initialization: $V(s) \leftarrow 0$ for all $s \in \mathcal{S}$
2: **for** $t = 1, 2, 3, \ldots$ **do**
3: $\quad V \leftarrow \mathfrak{B}V$
4: **end for**

3.1.2 Asynchronous Value Iteration

Value iteration is simple to implement and is quite efficient for problems with small state and action spaces. But in every iteration it has to sweep over the whole state–action space to update the value function, which renders it expensive in large-scale problems.

In practice, however, such a thorough sweep is not necessary, due to a few reasons. First, a state can be *unimportant* if selecting a wrong action in this state does not have large (negative) impact on the rewards received by the agent (see, *e.g.*, Li [2004] and Li et al. [2007]). Second, the accuracy of the value function in some states may not be important to a reward-maximizing agent if these states are unlikely to be visited or unreachable. Finally, if some states' values are already close to the optimal values, performing updates of their values will result in small changes in the value function (and possibly no change in the resulting greedy policy); in contrast, it may be more economic to perform updates in states whose values are less accurate.

The last two reasons motivate the algorithmic idea of updating some states' values in an adaptive way that is not necessarily determined beforehand. In contrast to the basic form given in Algorithm 1, which is often called synchronous value iteration, the many variants of asynchronous value iteration [Bertsekas 1982; Bertsekas and Tsitsiklis 1989] update the value function in an asynchronous manner. A basic form is given in Algorithm 2, where different variants adopt different strategies to select states for performing value backup. It is well known that if every state $s \in \mathcal{S}$ is chosen infinitely often for value update, then V_t converges to V^* [Bertsekas and Tsitsiklis 1989].

Algorithm 2 Asynchronous value iteration

0: **Inputs:** $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
1: Initialization: $V(s) \leftarrow 0$ for all $s \in \mathcal{S}$
2: **for** $t = 1, 2, 3, \ldots$ **do**
3: Choose a state $s_t \in \mathcal{S}$ and update its state-value estimate: $V(s_t) \leftarrow \mathfrak{B}V(s_t)$.
4: **end for**

In the real-time dynamic programming (RTDP) algorithm [Barto et al. 1995], for instance, the agent only performs a value-function backup in states it actually visits (*c.f.*, Algorithm 3), and thus can spend its limited computational resources on states that are more important, according to the second reason above. Although assumptions like ergodicity are usually needed to guarantee every state–action pair be chosen infinitely often for value update, online performance guarantees can be established for variants of RTDP based on the analytic tools developed in this dissertation (§8.1).

In contrast to the *online* version of asynchronous value iteration like RTDP, we only

Algorithm 3 A generic form of real-time dynamic programming.

Inputs: $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
Initialize $Q(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$.
Initialize the initial state $s_1 \in \mathcal{S}$.
for $t = 1, 2, 3, \ldots$ **do**
  Take some action $a_t \in \mathcal{A}$.
  Update the value function: $Q(s_t, a_t) \leftarrow \mathfrak{B}Q(s_t, a_t)$.
  Reach a next state $s_{t+1} \sim T(\cdot|s_t, a_t)$.
end for

focus on the *offline* version where the agent does not choose actions, but its pure goal is to compute a near-optimal value function. Specifically, we describe a class of algorithms known as prioritized sweeping that prioritizes states according to certain errors (for example, the Bellman error defined in Equation 2.14), which is motivated by the third reason above. This approach[1], originally proposed by Moore and Atkeson [1993] and Peng and Williams [1993], has been quite successful in practice and resulted in a number of variants [Andre et al. 1998; McMahan and Gordon 2005; Wingate and Seppi 2005]. These algorithms rely on a priority function, H, and update the values of states with highest priority. Algorithm 4 gives a generic form of prioritized-sweeping algorithms, which actually includes many specific instances of asynchronous value iteration with appropriate priority functions H. Again, we use subscript to denote the value of the quantity in the t-th iteration of Algorithm 4; for instance, H_t and V_t.

Algorithm 4 An abstract form of prioritized sweeping.

Inputs: $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
Initialize $V(s)$ for all $s \in \mathcal{S}$.
Initialize priority values $H(s)$ for all $s \in \mathcal{S}$.
for $t = 1, 2, 3, \cdots$ **do**
  Pick the state with the highest priority: $s_t \leftarrow \operatorname{argmax}_{s \in \mathcal{S}} H(s)$.
  Perform Bellman backup on state s_t: $V(s_t) \leftarrow \mathfrak{B}V(s_t)$.
  Update priority values for all $s \in \mathcal{S}$.
end for

1. Asynchronous value iteration given in Algorithm 2 repeatedly performs Bellman backups in an *arbitrary* state ordering, which is easily guaranteed by many $H(s)$

[1] We note that prioritized sweeping was first proposed as a reinforcement-learning algorithm rather than a planning algorithm.

functions. For example, let $S = \{1, 2, \cdots, n\}$ and suppose we wish to perform Bellman backups in the order of $1 \to 2 \to \cdots \to n-1 \to n \to 1 \to 2 \to \cdots$, then this is guaranteed by the following priority function: It is initialized for every state s by

$$H_1(s) \leftarrow \frac{2n - s}{2n} \tag{3.1}$$

and updated by

$$H_{t+1}(s) \leftarrow \begin{cases} H_t(s), & \text{if } s \neq s_t \\ \frac{H_t(s)}{2} & \text{if } s = s_t \end{cases}. \tag{3.2}$$

Therefore, asynchronous value iteration is a special case of Algorithm 4.

2. The PS algorithm [Moore and Atkeson 1993] does not specify how to initialize H. In practice, we may initialize $H(s)$ with random positive values.[2] The algorithm then updates $H(s)$ in the following way: for any state $s \in S$,

$$H_{t+1}(s) \leftarrow \begin{cases} \max\left\{H_t(s), \Delta_t \cdot \max_{a \in A} T(s_t|s,a)\right\}, & \text{if } s \neq s_t \\ \Delta_t \cdot \max_{a \in A} T(s_t|s,a), & \text{if } s = s_t \end{cases}. \tag{3.3}$$

where $\Delta_t = |V_{t+1}(s_t) - V_t(s_t)| = |E(s_t; V_t)|$ is the change of s_t's state value after the most recent Bellman backup.

3. The GenPS algorithm [Andre et al. 1998] updates $H(s)$ so that it is always the absolute Bellman error in state s. Specifically, for all states $s \in S$,

$$H_{t+1}(s) \leftarrow |E(s; V_{t+1})|. \tag{3.4}$$

Note that Andre et al. [1998] also provide a heuristic approach to updating $H(s)$ without explicitly computing Bellman errors in all states. We only consider a version of GenPS that always maintains the exact absolute Bellman errors.[3] As shown by Lemma 8 in the next subsection, this condition can be satisfied quite efficiently without recomputing Bellman errors for all states in every step.

---

[2] See Li and Littman [2008b] for an example that shows PS may not converge to the optimal value function if some $H(s)$ is initialized to 0.

[3] See Li and Littman [2008b] for an example that shows the heuristic GenPS algorithm may not converge to the optimal value function.

4. Since the IPS algorithm [McMahan and Gordon 2005] does not perform Bellman backups, it is quite different from all these Bellman-backup-based planning algorithms above, and thus is not straightforward to convert to the form of Algorithm 4. However, it becomes a special case of Algorithm 4 if $\mathcal{A}$ contains only one action; namely, if the MDP is actually a Markov chain. More specifically, IPS is for episodic problems in which:[4]

   - there is an absorbing goal state s_g so that $R(s_g, a) = 0$ for all $a \in A$, and
   - all other rewards are negative: $-1 \leq R(s, a) < 0$ for all $s \neq s_g$ and $a \in A$.

   Consequently, $V^*(s_g) = 0$ and there exists a constant $\nu \in (0, \frac{1}{1-\gamma})$ such that $V^*(s) \leq -\nu$ for all $s \in S \setminus \{s_g\}$; that is, all non-goal states' values differ from 0 by at least ν. This class of problems are also called *stochastic shortest path problems* [Bertsekas 2001]. IPS can be viewed as an instance of Algorithm 4:[5] it initializes the value function *pessimistically*

$$V_1(s) \leftarrow \begin{cases} -\frac{1}{1-\gamma} & \text{if } s \neq s_g \\ 0 & \text{otherwise,} \end{cases}$$

   initializes the priority value by

$$H_1(s) \leftarrow \begin{cases} 0 & \text{if } s \neq s_g \\ 1 & \text{otherwise,} \end{cases}$$

   and updates priority values according to

$$H_{t+1}(s) \leftarrow -\frac{E(s; V_{t+1})}{V_{t+1}(s) + E(s; V_{t+1}(s))}.$$

   It can be shown that the value function V_t will not decrease as the algorithm operates (*c.f.*, Lemma 9), and $H_t(s) \geq 0$ at all times (*c.f.*, Lemma 10). The motivation of the priority value above is that IPS reduces to the highly efficient Dijkstra's

[4] We have adapted the algorithm to be consistent with our notation and setting. The algorithm was originally proposed for non-discounting, minimum-cost problems by McMahan and Gordon [2005], while we use a discount factor and consider reward-maximizing problems.

[5] The formulation we present here is not optimal for implementation, but will suffice for our theoretical purposes in this paper.

algorithm when the Markov chain is a deterministic, acyclic graph [McMahan and Gordon 2005].

We now present convergence results that are recently established by Li and Littman [2008b]. Complete proofs are given in §3.4. The following key lemma gives a set of sufficient conditions for the convergence of Algorithm 4.

Lemma 7 *Let $\mathcal{F} \subseteq \mathcal{S}$ be the set of states that are chosen for Bellman backups infinitely often during the whole run of Algorithm 4; so, $\mathcal{S} \setminus \mathcal{F}$ consists of states that eventually become starved of backups. Let τ be the* last *step in which some state in $\mathcal{S} \setminus \mathcal{F}$ is chosen for update. Clearly, $\mathcal{F} \neq \emptyset$ by the Pigeonhole Principle. Then the Bellman errors of infinitely updated states are driven to* 0 *in the limit; formally,*

$$\lim_{t \to \infty} \max_{s \in \mathcal{F}} |E(s; V_t)| = 0. \tag{3.5}$$

Furthermore, V_t converges to V^* if the following conditions hold:

1. *The priority values converge to* 0 *in the limit:*

$$\lim_{t \to \infty} H_t(s_t) = 0.$$

2. *There exists a constant $C > 0$ such that: $H_t(s) \geq C \cdot |E(s; V_t)|$ for all states $s \notin \mathcal{F}$ and $t > \tau$;*

Some intuitions are helpful. The first condition requires that, in the limit, all priority values must approach 0 so that no state with a positive priority value will be starved of updates. The second condition requires that, in the limit, the priority value for starving states *must not be too small* compared to their absolute Bellman errors. Therefore, these two conditions together guarantee that any starving state must have a zero Bellman error, and thus does not need Bellman backups at all.

Lemma 7 allows one to prove a number of variants of prioritized-sweeping algorithms converge to the optimal value function in the limit:[6]

[6]The literature contains a number of references to prioritized sweeping, but the only informal argument for its asymptotic convergence to the optimal value function is flawed. Thus, although the community appears confident the algorithm converges, no published proof exists to the best of my knowledge.

Theorem 1 *GenPS of Andre et al. [1998] converges to the optimal value function in the limit.*

Theorem 2 *PS of Moore and Atkeson [1993] converges to the optimal value function in the limit, if the initial priority values are non-zero, namely,* $H_1(s) > 0$ *for all* $s \in S$.

Theorem 3 *IPS of McMahan and Gordon [2005] in a Markov chain converges to the optimal value function in the limit.*

3.1.3 Policy Iteration

Another classic dynamic-programming approach to planning in MDPs is policy iteration [Howard 1960]. This algorithm directly searches for the optimal policy in the policy space. Starting with an arbitrary policy, the algorithm proceeds in an iterative way: in every iteration, it first performs the *policy-evaluation* step by computing the value function of the current policy (which is equivalent to solving the system of linear equations in Equation 2.15 or 2.16), and then *improves* the policy by updating the policy to be the greedy policy with respect to the value function of the current policy. The pseudocode of policy iteration is given in Algorithm 5.

Algorithm 5 Policy iteration.

0: **Inputs:** $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
1: Initialize $\pi_1 \in \mathcal{A}^{\mathcal{S}}$ arbitrarily.
2: **for** $t = 1, 2, 3, \ldots$ **do**
3: Policy evaluation: solve for Q^{π_t}.
4: Policy improvement: define π_{t+1} that is greedy w.r.t. Q^{π_t}.
5: **end for**

It is known that the every policy π_t in policy iteration dominates all previous policies π_τ for $\tau < t$ [Puterman 1994, Proposition 6.4.1]. Since the number of deterministic policies is finite in finite MDPs ($|\mathcal{A}|^{|\mathcal{S}|}$), policy iteration is guaranteed to terminate after finitely many iterations. Although policy iteration often terminates in quite few iterations in practice, it remains a long standing open problem whether it terminates after a polynomial (in $|\mathcal{S}|$, $|\mathcal{A}|$, $\log \frac{1}{1-\gamma}$) number of steps. In fact, it is known that certain variants of policy iteration may take exponential time [Littman et al. 1995]. However,

policy iteration is a polynomial-time algorithm if γ is fixed or if γ is represented in unary [Littman et al. 1995]. Complexity results that are independent of γ are investigated by Mansour and Singh [1999], but they are exponential in $|\mathcal{S}|$. Interested readers are also referred to Madani [2002] for discount-independent complexity bounds for this algorithm in restricted classes of MDPs.

3.1.4 Linear Programming

In addition to dynamic programming, linear programming also provides a solution method for planning in finite MDPs [d'Epenoux 1963]. Specifically, the optimal value function of an MDP can be formulated as the solution to a linear program whose size is polynomial in the representation size of the MDP. Since linear programs are polynomially solvable, their relation to MDPs is currently the only proof that MDPs can be solved in polynomial time [Littman et al. 1995]. Given a finite MDP $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, the linear program can be defined as:

$$\min_{V \in \mathbb{R}^{|\mathcal{S}|}} \sum_{s \in \mathcal{S}} V(s) \qquad \text{subject to: } V \geq \mathfrak{B}V, \tag{3.6}$$

whose solution is the optimal value function, V^*. The optimization problem above is indeed a linear program. To see it, observe that the nonlinear constraint $V(s) \geq (\mathfrak{B}V)(s)$ can be turned into a system of $|\mathcal{A}|$ many linear inequalities:

$$V(s) \geq R(s) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s, a) V(s'), \quad \forall a \in \mathcal{A}.$$

Therefore, the inequality $V \geq \mathfrak{B}V$ is in fact a system of $|\mathcal{S}|\,|\mathcal{A}|$ many linear inequalities.

3.2 Compact Representation and Function Approximation

One of the most important benefits of using function approximation is to avoid representing the value function or policy by a lookup table. There are many alternatives such as using multi-layer neural networks [Tesauro 1995; Crites and Barto 1996], locally weighted regression [Atkeson et al. 1997a], decision trees [McCallum 1995; Uther 2002] , and Gaussian processes [Engel et al. 2003; Rasmussen and Kuss 2004], just to name a few. In the following, we only review two popular techniques known as state

abstraction and linear function approximation that will be useful in our discussions in the next chapters.

3.2.1 State Abstraction

State abstraction (*a.k.a. state aggregation*) has been extensively studied in artificial intelligence (*e.g.*, Giunchiglia and Walsh [1992]) and operations research (*e.g.*, Rogers et al. [1991]) as a technique for accelerating decision making. Abstraction can be thought of as a process that maps the *ground* representation—the original description of a problem—to an *abstract* representation, a much more compact and easier one to work with [Giunchiglia and Walsh 1992]. In other words, abstraction allows the agent to distinguish *relevant* information from *irrelevant* information. From a computational perspective, state abstraction is a technique for making learning and planning algorithms practical on large, real-world problems.

Here, we focus on state abstraction in Markov decision processes, where different types of abstraction have been proposed such as *bisimulation* [Givan et al. 2003], *homomorphism* [Ravindran and Barto 2003], *utile distinction* [McCallum 1995], and *policy irrelevance* [Jong and Stone 2005]. Despite many successes, negative results are reported when using state abstraction in MDPs. For example, McCallum [1995] has observed that aggregating states using one form of state abstraction makes it impossible to find the optimal policy using value iteration or Q-learning (*c.f.*, §4.1.1); Gordon [1996] reported a chattering phenomenon of Sarsa(λ) when combined with improperly constrained state abstraction.

These undesirable results raise important questions like "What information is lost when an abstraction is applied?" and "When is the optimal policy still preserved?". This concern motivates a unified theory of state abstraction proposed by Li et al. [2006], which we will summarize below.

Giunchiglia and Walsh [1992] argue that abstraction is in general *a mapping from one problem representation to a new representation, while preserving certain properties.* Here, we focus on the preservation of properties that are needed for an agent to find an optimal policy in an MDP. Previous abstraction definitions have applied this insight.

For example, bisimulation [Givan et al. 2003] is essentially a type of abstraction that preserves the one-step dynamics of an MDP (*i.e.*, the transition and reward functions), while policy irrelevance abstraction attempts to preserve the optimal actions [Jong and Stone 2005]. Many of the methods we discussed attempt "fuzzy conservation" of such properties through statistical tests and notions of bounding, but in the interest of developing a formalism we choose to focus on abstraction schemes where states are only aggregated when they have exact equality over the parameters of the abstraction scheme.

Definition 4 *Let $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ be the* ground *MDP and its* abstract *version be $\bar{M} = \langle \bar{\mathcal{S}}, \mathcal{A}, \bar{T}, \bar{R}, \gamma \rangle$. Define the* abstraction function *as $\phi \in \bar{\mathcal{S}}^{\mathcal{S}}$; $\phi(s) \in \bar{\mathcal{S}}$ is the abstract state corresponding to ground state s, and the* inverse image *$\phi^{-1}(\bar{s})$, with $\bar{s} \in \bar{\mathcal{S}}$, is the set of ground states that correspond to $\bar{s}$ under abstraction function ϕ. Note that under these assumptions, $\{\phi^{-1}(\bar{s}) \mid \bar{s} \in \bar{\mathcal{S}}\}$ partitions the ground state space $\mathcal{S}$. To guarantee $\bar{T}$ and $\bar{R}$ are well-defined, a weighting function is needed: $w \in [0,1]^{\mathcal{S}}$, such that for each $\bar{s} \in \bar{\mathcal{S}}$, $\sum_{s \in \phi^{-1}(\bar{s})} w(s) = 1$. With these definitions at hand, we can define the transition and reward functions of the abstract MDP as follows:*

$$\begin{aligned} \bar{R}(\bar{s}, a) &\stackrel{\text{def}}{=} \sum_{s \in \phi^{-1}(\bar{s})} w(s) R(s, a), \\ \bar{T}(\bar{s}' \mid \bar{s}, a) &\stackrel{\text{def}}{=} \sum_{s \in \phi^{-1}(\bar{s})} \sum_{s' \in \phi^{-1}(\bar{s})} w(s) T(s' \mid s, a). \end{aligned}$$

It can be verified that $\bar{T}(\bar{s}' \mid \bar{s}, a)$ is a well-defined next state distribution: for any $\bar{s} \in \bar{\mathcal{S}}$ and $a \in \mathcal{A}$:

$$\begin{aligned} \sum_{\bar{s}' \in \bar{\mathcal{S}}} \bar{T}(\bar{s}' \mid \bar{s}, a) &= \sum_{\bar{s}' \in \bar{\mathcal{S}}} \sum_{s \in \phi^{-1}(\bar{s})} \sum_{s' \in \phi^{-1}(\bar{s})} w(s) T(s' \mid s, a) \\ &= \sum_{s \in \phi^{-1}(\bar{s})} \sum_{\bar{s}' \in \bar{\mathcal{S}}} \sum_{s' \in \phi^{-1}(\bar{s})} w(s) T(s' \mid s, a) \\ &= \sum_{s \in \phi^{-1}(\bar{s})} w(s) \sum_{s' \in \mathcal{S}} T(s' \mid s, a) \\ &= \sum_{s \in \phi^{-1}(\bar{s})} w(s) \;=\; 1. \end{aligned}$$

Intuitively, $w(s)$ measures the extent to which state s contributes to the abstract state $\phi(s)$. In the rest of the paper, we will only mention w when necessary; otherwise, it can

be any valid weighting function. We now consider how policies $\bar{\pi}$ in the abstract MDP translate to policies π in the ground MDP. Since all ground states in $\phi^{-1}(s)$ are treated identically, it is natural to translate policies by the following rule: $\pi(s,a) = \bar{\pi}(\phi(s),a)$ for all s and a. Finally, value functions for the abstract MDP $\bar{M}$ can be defined in the straightforward way and are denoted $V^{\bar{\pi}}(\bar{s})$, $V^*(\bar{s})$, $Q^{\bar{\pi}}(\bar{s},a)$, and $Q^*(\bar{s},a)$, respectively.

We are now ready to define five types of state abstraction in MDPs and categorize much previous work using this language.

Definition 5 *Given an MDP $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, and any states $s_1, s_2 \in \mathcal{S}$, we define five types of abstraction as below, with an arbitrary but fixed weighting function $w(s)$.*[7]

1. *A* model-irrelevance abstraction ϕ_{model} *is such that for any action a and any abstract state $\bar{s}$, $\phi_{\text{model}}(s_1) = \phi_{\text{model}}(s_2)$ implies*

$$R(s_1,a) = R(s_2,a) \quad \text{and} \quad \sum_{s' \in \phi^{-1}_{\text{model}}(\bar{s})} T(s' \mid s_1, a) = \sum_{s' \in \phi^{-1}_{\text{model}}(\bar{s})} T(s' \mid s_2, a).$$

2. *A* Q^π-irrelevance abstraction ϕ_{Q^π} *is such that for any policy π and any action a, $\phi_{Q^\pi}(s_1) = \phi_{Q^\pi}(s_2)$ implies $Q^\pi(s_1,a) = Q^\pi(s_2,a)$.*
3. *A* Q^*-irrelevance abstraction ϕ_{Q^*} *is such that for any action a, $\phi_{Q^*}(s_1) = \phi_{Q^*}(s_2)$ implies $Q^*(s_1,a) = Q^*(s_2,a)$.*
4. *An* a^*-irrelevance abstraction ϕ_{a^*} *is such that every abstract class has an action a^* that is optimal for all the states in that class, and $\phi_{a^*}(s_1) = \phi_{a^*}(s_2)$ implies that $Q^*(s_1,a^*) = \max_a Q^*(s_1,a) = \max_a Q^*(s_2,a) = Q^*(s_2,a^*)$.*
5. *A* π^*-irrelevance abstraction ϕ_{π^*} *is such that every abstract class has an action a^* that is optimal for all the states in that class, that is $\phi_{\pi^*}(s_1) = \phi_{\pi^*}(s_2)$ implies that $Q^*(s_1,a^*) = \max_a Q^*(s_1,a)$ and $Q^*(s_2,a^*) = \max_a Q^*(s_2,a)$.*

Intuitively, ϕ_{model} preserves the one-step model (*e.g.*, bisimulation [Givan et al. 2003]); ϕ_{Q^π} preserves the state–action value function for all policies; ϕ_{Q^*} preserves the optimal state–action value function (*e.g.*, stochastic dynamic programming with factored representations [Boutilier et al. 2000] or the G-algorithm [Chapman and Kaelbling

[7] *Although w does not appear important in the definition, it can play an important role in affecting planning and/or learning efficiency [Van Roy 2006].*

1991]); ϕ_{a^*} preserves the optimal action and its value (*e.g.*, utile distinction [McCallum 1995]); and ϕ_{π^*} attempts to preserve the optimal action [Jong and Stone 2005].

It is shown by Li et al. [2006] that by using any of the four abstractions: ϕ_{model}, ϕ_{Q^π}, ϕ_{Q^*}, and ϕ_{a^*}, the optimal policy in the abstract MDP will be optimal in the ground MDP; however, abstraction ϕ_{π^*} might lead to a suboptimal policy in the ground MDP.[8] Similar results hold for reinforcement learning.[9]

Before ending this subsection, we note again that the abstractions in Definition 5 are based on exact equivalence of certain quantities. Although this definition has been useful for developing a formal framework to study properties of state abstraction, it is often too stringent in practice, especially in stochastic domains and discounted MDPs. One possible extension is to relax the exactness and allow some form of approximate or soft abstraction [Bertsekas and Castañon 1989; Dean et al. 1997; Even-Dar and Mansour 2003a; Ferns et al. 2004; Li and Littman 2005; Van Roy 2006; Taylor et al. 2009].

3.2.2 Linear Function Approximation

Linear function approximation is another general representation scheme. It is widely used in planning (*e.g.*, Schweitzer and Seidmann [1985]), reinforcement learning (*e.g.*, Samuel [1959] and Sutton [1988]), and machine learning (*e.g.*, Hastie et al. [2003]). It is useful for at least a few reasons. First, the class of linear functions often provide good approximations to the target function when the set of features (see below) is reasonably well selected. Furthermore, the kernel trick [Shawe-Taylor and Cristianini 2004] can be employed to boost the representational power of linear approximation schemes. Second, the linear approximation scheme is relatively simple compared to nonlinear schemes like neural networks, rendering intuitive understanding and rigorous analysis of its behavior easier. Finally, linear approximation appears stabler than nonlinear approximation when they are used in reinforcement learning; see §4.2.2 for more details.

---

[8] Related counterexamples for the suboptimality led by ϕ_{a^*} are given by McCallum [1995] and Jong and Stone [2005].

[9] When ϕ_{a^*} is used, however, a chattering behavior has been observed for some reinforcement-learning algorithms like Sarsa [Gordon 1996; Li et al. 2006]. But this kind of chattering does not affect the optimality of the greedy policy Sarsa converges to.

Given a set of k *basis functions* (*a.k.a. features*):

$$\phi_i \in \mathbb{R}^{\mathcal{X}}, \quad i = 1, 2, \ldots, k$$

where $\mathcal{X}$ is the set of inputs, we may use a linear combination of them, denoted $\hat{f}$, to approximate a target function $f : \mathcal{X} \to \mathbb{R}$. That is, we seek a vector $\mathbf{w} = [w_1, w_2, \ldots, w_k]^\top \in \mathbb{R}^k$ so that

$$\hat{f} \stackrel{\text{def}}{=} \sum_{i=1}^{k} w_i \phi_i \approx f.$$

For convenience, we denote the feature vector by

$$\boldsymbol{\phi} \stackrel{\text{def}}{=} [\phi_1, \phi_2, \ldots, \phi_k]^\top \in \left(\mathbb{R}^k\right)^{\mathcal{X}},$$

and so the linear approximation is compacted written as $\hat{f} = \mathbf{w}^\top \boldsymbol{\phi}$.

There are various ways to define optimal weight vectors in the search of good linear approximations. For instance, the linear least-squares solution is a linear approximation that minimizes squared differences between $\mathbf{w}^\top \boldsymbol{\phi}$ and f:

$$\mathbf{w}^* \stackrel{\text{def}}{=} \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^k} \left\| \mathbf{w}^\top \boldsymbol{\phi} - f \right\|_2^2.$$

When the set of basis functions, $\{\phi_1, \phi_2, \ldots, \phi_k\}$, is linearly independent, the least-squares solution $\mathbf{w}^*$ is unique.

Linear approximation is quite general, including a number of important cases with appropriately chosen features. When we use the *indicator feature* for a finite input set, we can recover the tabular representation, in which each component in the weight vector corresponds to an entry in the lookup table. Similarly, linear approximation also subsumes the abstraction/aggregation technique discussed in the previous subsection, provided that the number of aggregated inputs is finite. Finally, we note that a wide class of function approxitors known as *averagers* (*c.f.*, §3.3.2) can be interpreted as a special kind of linear approximator, including k-nearest neighbor regression and kernel regression among others.

We now discuss a few popular choices of basis functions when the input space $\mathcal{X}$ is a subset of $\mathbb{R}^n$, and every input $\mathbf{x} \in \mathcal{X}$ is represented as a column vector: $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$.

- A *polynomial basis function* ϕ of degree d can be represented by:

$$\phi(\mathbf{x}) = \prod_{i=1}^{n} x_i^{d_i}, \quad s.t. \quad d_i \in \mathbb{Z}_+ \text{ and } \sum_{i=1}^{n} d_i = d.$$

  For example, a degree-0 polynomial basis is the constant function, and a degree-1 polynomial basis can be one of the n possibilities: $x_1, x_2, \ldots, x_n$. It is known that the set of all polynomial basis functions of degree up to d contains $\binom{n+d}{d}$ functions, and that they are linearly independent [Cheney and Light 2000, Chapter 4]. A nice property of polynomial bases is the Weierstrass approximation theorem, which states that any continuous function f defined in a bounded and closed set $\mathcal{X} \subset \mathbb{R}^n$ can be approximated to arbitrary precision by polynomial functions. However, using high order polynomial basis functions can be numerically unstable.

- *Radial basis functions (RBFs)* define features for inputs based on a set of fixed prototypical inputs. Specifically, let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ be k *fixed*, *distinct* points (called "centers") in $\mathcal{X}$. For any $\mathbf{x} \in \mathcal{X}$, we can define k basis functions by:[10]

$$\phi_i(\mathbf{x}) \stackrel{\text{def}}{=} \exp\left(-\tau \left\|\mathbf{x} - \mathbf{x}_i\right\|^2\right).$$

  where $\tau \in \mathbb{R}_+$ is an adjustable constant, $\|\cdot\|$ is any metric defined on $\mathcal{X}$, and $i = 1, 2, \ldots, k$. When the k centers are distinct, then the set of radial basis functions defined above is linear independent [Cheney and Light 2000, Chapter 15]. RBFs are *local* basis functions in the sense that each basis $\phi_i(\mathbf{x})$ achieves maximum value at $\mathbf{x}_i$ and vanishes to 0 when $\mathbf{x}$ is far away from $\mathbf{x}_i$. Hence, any change in the corresponding coefficient, w_i, has a diminishing effect on the approximation in regions far away from $\mathbf{x}_i$. The parameter τ controls the speed of diminishing. In contrast, polynomial basis functions are *global*.

- *Cerebellar model articulation controller* or *CMAC* is a type of neural networks motivated by neuroscience [Albus 1971]. It is also called *cerebellar model arithmetic computer* [Miller et al. 1990] or *tile coding* [Sutton and Barto 1998]. CMAC may be viewed a multi-layer discretization. At every layer, it poses a grid on the

[10] Note that a more general definition of RBFs is possible. Interested readers are referred to Chapter 15 of the textbook by Cheney and Light [2000].

input space $\mathcal{X}$ to divide it into a number N of hyper-rectangles, each of which corresponds to a cell in the grid. The grids for different layers are different. At layer i, each input point $\mathbf{x} \in \mathcal{X}$ has a feature vector $\boldsymbol{\phi}_i \in \{0,1\}^N$ consisting of zeros except in the position corresonding to the cell in the grid where $\mathbf{x}$ is located. The final feature vector in CMAC is a concatenation of these vectors:

$$\boldsymbol{\phi}(\mathbf{x}) = \left[\boldsymbol{\phi}_1(\mathbf{x})^\top, \boldsymbol{\phi}_2(\mathbf{x})^\top, \cdots, \boldsymbol{\phi}_K(\mathbf{x})^\top\right]^\top,$$

where K is the number of layers in CMAC. If every layer of discretization divides $\mathcal{X}$ into N cells, then $\boldsymbol{\phi}(\mathbf{x}) \in \{0,1\}^{NK}$; furthermore, $\boldsymbol{\phi}(\mathbf{x})$ contains exactly K ones for all $\mathbf{x}$. CMAC then approximates a target function using these $k = NK$ basis functions in CMAC to do linear approximation. CMAC has been popular and successful in many reinforcement-learning systems (see, *e.g.*, Miller et al. [1990] and Sutton and Barto [1998]).

In addition to those surveyed above, other pre-defined features like proto-value functions [Mahadevan and Maggioni 2007], Krylov basis function [Petrik 2007], and Fourier basis functions [Konidaris and Osentoski 2008] are frequently used in practice. But they may not always be effective for all problems. If the feature set is not expressive enough to capture the underlying value function, a linear value-function approximation will likely lead to a poor policy no matter what learning algorithm is used. On the other hand, using a large pool of features helps avoid this problem, but at the cost of increased computational and sample complexity.

The applicability of linear function approximation can be augmented by feature-selection procedures like matching pursuit [Mallat and Zhang 1993] and Lasso [Tibshirani 1996]. This problem is an extremely important question that we cannot cover in this dissertation. In the context of MDP planning and learning, interesting progress has been made recently for feature section, including the use of Bellman errors [Menache et al. 2005; Keller et al. 2006; Parr et al. 2007], spectral analysis [Mahadevan and Maggioni 2007], Lasso-type regularization [Loth et al. 2007; Kolter and Ng 2009b], and others [Parr et al. 2008; Li et al. 2009c].

3.3 Approximate Planning in Large Markov Decision Processes

Although MDPs can be solved exactly in polynomial time (*c.f.*, §3.1.4), most MDPs of practical interest are too large to solve exactly. For instance, in the game of 19×19 Go, the total number of states (legal board positions) is estimated to be 2.08×10^{170}, which far exceeds the number of atoms in the known universe [Tromp and Farnebäck 2007]. Consequently, it is impossible even to represent the Q-function for all possible state by a lookup table (as what we assumed in §3.1), let alone running the exact planning algorithms such as value iteration.[11] As another example, many real-life problems such as robotics applications exhibit continuous state spaces that contain infinitely many states, which makes approximation necessary unless restricted assumptions are made about the dynamics of the problem. This section describes *approximation* algorithms that are feasible for large-scale problems such as Go at the price of losing the guarantee of computing the exact optimal value functions.

There have been a number of approximate planning algorithms. Most of them rely on compact representations of the value functions based on techniques surveyed in §3.2 as opposed to the table-based representation. Below we first review some of the most popular function approximation architectures, and then survey a few approximate planning algorithms. Other approximate approaches are mentioned in the end.

3.3.1 State Abstraction

A natural way to solve a large MDP M is to solve an abstract MDP $\bar{M}$, and then use the optimal policy in $\bar{M}$ as a (hopefully) near-optimal policy in the original, ground MDP. Such a method has success in solving large MDPs with performance guarantees [Givan et al. 2000; Tsitsiklis and Van Roy 1996; Van Roy 2006; Taylor et al. 2009].

State abstraction in solving continuous MDPs is also known as aggregation or discretization. In this context, the state and action spaces are often assumed to be multidimensional cubes: $\mathcal{S} = [0,1]^n$ and $\mathcal{A} = [0,1]^m$ for some $n, m \in \mathbb{N}$. Discretizing the

[11] As a two-player game, Go has the additional complexity that makes it $\mathcal{EXP}$-complete [Robson 1983].

state space *evenly* with *resolution* $h \in (0,1)$ results in a $\frac{1}{h} \times \frac{1}{h} \times \cdots \frac{1}{h}$ grid with $\left(\frac{1}{h}\right)^n$ cells, and similar for the action space. A discretization algorithm first computes such finite partitions of the state and action spaces, then constructs an approximate MDP $\bar{M} = \langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{T}, \bar{R}, \gamma \rangle$ where $\bar{\mathcal{S}}$ and $\bar{\mathcal{A}}$ correspond to cells in the grids, and $\bar{P}$ and $\bar{R}$ are defined in different ways depending on the algorithm, and finally solves $\bar{M}$ using finite MDP planning algorithms surveyed in §3.1. Obviously, the optimal policy in $\bar{M}$ may be arbitrarily poor in the original MDP if no assumptions are made to limit the amount of information lost during discretization. Often, one has to assume that the transition and reward functions are sufficiently smooth (*i.e.*, *Lipschitz's continuity conditions*), and that the transition probability density function is bounded.

Under such assumptions, Bertsekas [1975] show that as the discretization becomes finer the resulting optimal value function in $\bar{M}$ become closer to the optimal value function in M; in the limit, they coincide. Later on, Chow and Tsitsiklis [1991] give a one-way multigrid algorithm whose computational complexity turns out to be (near) optimal [Chow and Tsitsiklis 1989]. For example, when the transition and reward functions are Lipschitz continuous and when the transition probability density is bounded, their algorithm is able to compute an approximate value function $\hat{V}$ such that $\left\| \hat{V} - V^* \right\|_\infty \leq \epsilon$ with computational complexity:

$$\mathcal{O}\left(\frac{1}{(1-\gamma)\left((1-\gamma)^2\epsilon\right)^{2n+m}} \right),$$

where the corresponding lower bound is

$$\Omega\left(\frac{1}{\left((1-\gamma)^2\epsilon\right)^{2n+m}} \right).$$

The process above discretizes the state and action spaces uniformly. In situations where the value function is flat in some regions, it is better to use *adaptive discretization* so that the same amount of computational resources (as measured by the total number of discretization cells) can be used to produce more accurate solutions [Bertsekas and Castañon 1989; Moore and Atkeson 1995; Munos and Moore 2002; Li and Littman 2005].

3.3.2 Approximate Dynamic Programming

Using dynamic programming in an abstract MDP in the previous subsection is in fact a special case of the more general method of *approximate dynamic programming*.

Approximate Value Iteration

Approximate value iteration (AVI) computes an estimate of the optimal value function by repeatedly applying an *approximate* Bellman operator to the present value function estimate, as opposed to applying the *exact* Bellman operator in value iteration. Starting from an initial value function V_1, AVI iterates using

$$V_{t+1} \leftarrow \hat{\mathfrak{B}} V_t,$$

where $t = 1, 2, \ldots$, and $\hat{\mathfrak{B}} \in \left(\mathbb{R}^{\mathcal{S}}\right)^{\mathbb{R}^{\mathcal{S}}}$ is an approximate Bellman operator. If $\hat{\mathfrak{B}}$ is a contraction such as in discretization algorithms, then V_t converges to its fixed point, denoted $\hat{V}^*$, that satisfies the Bellman equation: $\hat{V}^* = \hat{\mathfrak{B}}\hat{V}^*$. If $\hat{\mathfrak{B}}$ is close to $\mathfrak{B}$, then $\hat{V}^*$ is closed to V^*; formally (see, *e.g.*, Munos and Moore [2000]),

$$\left\|\hat{V}^* - V^*\right\|_\infty \leq \frac{1}{1-\gamma}\left\|\hat{\mathfrak{B}}V^* - V^*\right\|_\infty = \frac{1}{1-\gamma}\left\|\hat{\mathfrak{B}}V^* - \mathfrak{B}V^*\right\|_\infty.$$

Hence, the right-hand side becomes 0 if $\hat{\mathfrak{B}} = \mathfrak{B}$, which in turn implies $\hat{V}^* = V^*$.

A popular form of approximate value iteration is fitted value iteration [Gordon 1995], or FVI for short, where the approximate Bellman operator is the concatenation of the exact Bellman operator and an approximation operator: $\hat{\mathfrak{B}} = \mathfrak{A}\mathfrak{B}$, where $\mathfrak{A} \in \left(\mathbb{R}^{\mathcal{S}}\right)^{\mathbb{R}^{\mathcal{S}}}$ is an approximation operator. Therefore, the update rule of FVI, $V_{t+1} \leftarrow \mathfrak{A}\mathfrak{B}V_t$, can be viewed as a two-phase operation: the first phase involves the normal Bellman backup $\mathfrak{B}$ on the current value function V_t, and we denote the outcome function by $\hat{V}_t$; the second phase uses $\mathfrak{A}$ to approximate $\hat{V}_t$ in some parametric/succinct way and the outcome is V_{t+1}. $\mathfrak{A}$ can be an interpolation operator or, more generally, a supervised-learning algorithm:

- In lazy approximation [Li and Littman 2005], $\mathfrak{A}$ produces a piecewise constant approximation of $\hat{V}_t$ in a continuous MDP.

- In least-squares value iteration [Schweitzer and Seidmann 1985; Tsitsiklis and Van Roy 1996; Bertsekas and Tsitsiklis 1996], $\mathfrak{A}$ is a least-squares projection onto a space of functions linear in pre-defined features.
- In a randomized algorithm by Rust [1997], Monte Carlo techniques are used to replace the expensive Bellman operator by inexpensive sampling of next states.
- In smooth value iteration [Boyan and Moore 1995], $\mathfrak{A}$ is a supervised-learning algorithm such as locally weighted regression [Atkeson et al. 1997b] or backpropagation for neural networks [Mitchell 1997].

Unfortunately, the combination of $\mathfrak{A}$ and $\mathfrak{B}$ may not always be stable: although $\mathfrak{B}$ is a γ-contraction in the ℓ_∞-norm, as guaranteed in Lemma 5, the composition $\hat{\mathfrak{B}} = \mathfrak{A}\mathfrak{B}$ may be an expansion, potentially causing the sequence $[V_t]_{t\in\mathbb{N}}$ to diverge or oscillate in FVI. Such undesirable phenomena can be observed in practice for natural problems and popular choices of $\mathfrak{A}$ [Boyan and Moore 1995; Tsitsiklis and Van Roy 1996].

On the other hand, convergence is guaranteed if $\mathfrak{A}\mathfrak{B}$ is also a contraction. An important special case is when $\mathfrak{A}$ is a non-expansion, such as the so-called *averager* [Gordon 1995]; that is,

$$\|\mathfrak{A}V_1 - \mathfrak{A}V_2\|_\infty \leq \|V_1 - V_2\|_\infty .$$

Asymptotic as well as finite-time convergence rates are established under further assumptions about $\mathfrak{A}$ and $\mathfrak{B}$ [Munos 2007; Munos and Szepesvári 2008].

Approximate Policy Iteration

We can also obtain approximate versions of policy iteration. In Algorithm 5, two steps are involved: policy evaluation and policy improvement. In approximate policy iteration (API), both steps may be computed approximately. The reason we find approximate policy improvement useful is that the action space may be too large (*e.g.*, infinite) to compute the exact greedy action. For finite-action MDPs, policy improvement can always be performed exactly in time $\mathcal{O}(|\mathcal{A}|)$. For concreteness in our discussion, pseudocode of API is provided in Algorithm 6.

Unlike policy iteration, Algorithm 6 may not converge: due to the approximation in Lines 3 and 4, monotonicity in the sequence of policies, $[\pi_t]_{t\in\mathbb{N}}$, is not guaranteed except

Algorithm 6 Approximate policy iteration.

0: **Inputs:** $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
1: Initialize $\pi_1 \in \mathcal{A}^{\mathcal{S}}$ arbitrarily
2: **for** $t = 1, 2, 3, \ldots$ **do**
3: Approximate policy evaluation: compute $\hat{Q}^{\pi_t} \approx Q^{\pi_t}$.
4: Approximate policy improvement: compute $\pi_{t+1} \approx \operatorname{argmax} \hat{Q}^{\pi_t}$.
5: **end for**

in special cases (*e.g.*, Perkins and Precup [2003]). However, if the approximations in API are sufficiently accurate, the sequence of policies does converge to a near-optimal policy in a weaker sense. Define the policy-evaluation error by

$$\epsilon_{\mathrm{E}} \stackrel{\mathrm{def}}{=} \max_t \left\| \hat{Q}_t - Q^{\pi_t} \right\|_\infty$$

and the policy-improvement error by

$$\epsilon_{\mathrm{I}} \stackrel{\mathrm{def}}{=} \max_t \left\| Q^{\pi_{t+1}} - \mathfrak{B} Q^{\pi_t} \right\|_\infty ,$$

then the sequence of policies, $[\pi_t]_{t \in \mathbb{N}}$, approximates the optimal policy in the limit for finite MDPs in the sense that [Bertsekas and Tsitsiklis 1996]

$$\limsup_{t \to \infty} \left\| V^{\pi_t} - V^* \right\|_\infty \leq \frac{2\gamma\epsilon_{\mathrm{E}} + \epsilon_{\mathrm{I}}}{(1-\gamma)^2}.$$

This bound is limited partly because of the ℓ_∞-norm it uses as many approximation operators $\mathfrak{A}$ minimize function approximation errors in other norms. Extensions of this error bound have been established recently to weighted ℓ_p-norms [Munos 2003; Antos et al. 2008].

3.3.3 Approximate Linear Programming

Approximate linear programming (ALP) introduces approximation of value functions to the linear-programming formulation in Equation 3.6 for finite MDPs [de Farias and Van Roy 2003]. First, we use a linear function approximation (*c.f.*, §3.2.2): $V = \Phi\mathbf{w}$, where $\Phi \in \mathbb{R}^{|\mathcal{S}| \times k}$ is the *design matrix*, each row of which encodes the transposed feature vector for that state, and $\mathbf{w} \in \mathbb{R}^k$ is a weight vector. Then, we rewrite the linear program in Equation 3.6 with this approximation:

$$\min_{V \in \mathbb{R}^{|\mathcal{S}|}} \mathbf{c}^\top \Phi\mathbf{w} \qquad \text{subject to: } \Phi\mathbf{w} \geq \mathfrak{B}\Phi\mathbf{w}, \tag{3.7}$$

where $\mathbf{c} \in \mathbb{R}^k_+$ measures relative importance of each state in the objective of Equation 3.7. Similar to the constraint in Equation 3.6, the constraint above can be turned into a system of $|\mathcal{S}|\,|\mathcal{A}|$ many inequalities. Although ALP reduces the number of variables from $|\mathcal{S}|$ to k, the number of constraints is still too large as we try to avoid heavy dependence on $|\mathcal{S}|$ or $|\mathcal{A}|$. To reduce dependence on the size of the state and action spaces, constraint sampling can be used to solve Equation 3.7 approximately [de Farias and Van Roy 2004].

3.3.4 Sparse Sampling

Algorithms in the previous three subsections are all *global* methods: once the algorithm terminates, it produces a value function (in the form of a piecewise constant function, a linear function, or a trained neural network, *etc.*) that can be used to derive a greedy policy in *all* states using Equations 2.4 or 2.5. In contrast, *local* planning algorithms produce an action that is (near-)optimal only in *given* states.

An example is lookahead (heuristic) search algorithms [Russell and Norvig 2002] that are sometimes effective for solving deterministic MDPs. Conceptually, these methods create a search tree rooted at the state of interest by some form of forward search (depth-first search, breadth-first search, *etc.*) in the state space. If heuristic functions are available, there are techniques to prune the search tree to reduce computational complexity. In stochastic MDPs, however, the forward-search procedure above must be replaced by repeatedly sampling trajectories to re-construct the expected reward function and transition probabilities in reachable states. This idea appears in the sparse sampling algorithms [Kearns et al. 2002]. A similar technique known as rollout has also been useful [Tesauro and Galperin 1997].

Thanks to the discount factor (*c.f.*, Lemma 1), the depth of sparse sampling is on the order of $\tilde{\mathcal{O}}(1/(1-\gamma))$, making the algorithm's complexity independent of the number of states. However, the number of nodes in the forward search tree has an exponential dependence on the tree height, $\tilde{\mathcal{O}}(1/(1-\gamma))$. More recently, adaptive sparse sampling algorithms based on bandit exploration techniques (*c.f.*, §4.3.1) have been developed to reduce the computational complexity by reducing the amount of sampling [Chang

et al. 2005; Kocsis and Szepesvári 2006]. These algorithms have recently achieved considerable success in the challenging game of Go [Gelly et al. 2006].

3.4 Proofs

This section provides detailed proofs of all technical lemmas used in this chapter.

3.4.1 Proof of Lemma 7

The following lemma will also be useful in our convergence proof later. For convenience, define $E^a(s;V)$ for all $a \in \mathcal{A}$ as follows:

$$E^a(s;V) \stackrel{\text{def}}{=} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a)V(s') - V(s).$$

It follows from the definition that

$$E(s;V) = \max_{a \in \mathcal{A}} E^a(s;V). \tag{3.8}$$

As before, we will use V_t to denote value of V in the t-th iteration.

Lemma 8 *Using the notation in Algorithm 4, we have for all $t = 1, 2, \ldots$ that:*

$$E(s;V_{t+1}) = \begin{cases} \max_{a \in \mathcal{A}} \left[E^a(s;V_t) + \gamma T(s_t \mid s,a) E(s_t;V_t) \right], & \text{if } s \neq s_t \\ \max_{a \in \mathcal{A}} \left[E^a(s;V_t) + \gamma T(s_t \mid s_t,a) E(s_t;V_t) \right] - E(s_t;V_t), & \text{if } s = s_t. \end{cases} \tag{3.9}$$

Proof. We consider the two cases separately. For $s \neq s_t$ and any $a \in \mathcal{A}$,

$$\begin{aligned} E^a(s;V_{t+1}) &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) V_{t+1}(s') - V_{t+1}(s) \\ &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) V_t(s') + \gamma T(s_t \mid s,a) E(s_t;V_t) - V_t(s) \\ &= E^a(s;V_t) + \gamma T(s_t \mid s,a) E(s_t;V_t), \end{aligned}$$

where the first equality is due to the definition of E^a, the second equality follows from the fact that $V_{t+1}(s')$ differs from $V_t(s')$ only when $s' = s_t$ and the difference is exactly $E(s_t;V_t)$, and the last equality is again due to the definition of E^a. Using Equation 3.8,

we have proved the first part of Equation 3.9. For $s = s_t$ and any $a \in \mathcal{A}$, a similar reasoning follows:

$$\begin{aligned} E^a(s_t; V_{t+1}) &= R(s_t, a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s_t, a) V_{t+1}(s') - V_{t+1}(s_t) \\ &= R(s_t, a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s_t, a) V_t(s') + \gamma T(s_t \mid s_t, a) E(s_t; V_t) - \big(V_t(s_t) + E(s_t; V_t)\big) \\ &= E^a(s_t; V_t) + \gamma T(s_t \mid s_t, a) E(s_t; V_t) - E(s_t; V_t). \end{aligned}$$

Using Equation 3.8 again, we have proved the second part of Equation 3.9. □

Proof (of Lemma 7). Given any MDP $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, we run Algorithm 4 forever using an arbitrary bounded initial value function V_1. Now, we will construct an induced MDP $M' = \langle \mathcal{S}, \mathcal{A}, T', R', \gamma \rangle$ with the same states, actions, and discount factor. For states $s \in \mathcal{F}$, the reward function and transition probabilities are the same as M; for states $s \notin \mathcal{F}$ and any $a \in \mathcal{A}$,

$$R'(s, a) \stackrel{\text{def}}{=} V_{\tau+1}(s) \cdot (1 - \gamma) \quad \text{and} \quad T'(s' \mid s, a) \stackrel{\text{def}}{=} \mathbb{I}(s' = s).$$

In other words, we turn states $s \notin \mathcal{F}$ to absorbing states with self loops, and the new reward is constructed so that $V_{\tau+1}(s)$ is exactly the optimal value of s in the induced MDP M'.

After timestep τ, Algorithm 4 running on M can be viewed as running on M' since states $s \notin \mathcal{F}$ will not get their values updated anyway. By construction of M', the value function over these states converges to the optimal value function of M' [Bertsekas and Tsitsiklis 1989], since all states $s \in \mathcal{F}$ are updated infinitely often and all states $s \notin \mathcal{F}$ already have their optimal values. Hence, the Bellman error evaluated using the model of M' converges to 0 for all states $s \in \mathcal{S}$.

Finally, we consider the Bellman error evaluated using the model of M. For states $s \in \mathcal{F}$, the Bellman error in s converges to 0, as M and M' have the same dynamics for these states. Thus, we have proved the first part of the lemma:

$$\lim_{t \to \infty} \max_{s \in \mathcal{F}} |E(s; V_t)| = 0.$$

We now consider Bellman errors of states outside of $\mathcal{F}$. For any $t > \tau$, we have

$$\max_{s \notin \mathcal{F}} |E(s; V_t)| \;\le\; \max_{s \notin \mathcal{F}} \frac{H_t(s)}{C} \;\le\; \frac{H_t(s_t)}{C},$$

where the first inequality follows from Condition 2, and the second from the operations of Algorithm 4. But, Condition 1 requires that $H_t(s_t)$ converges to 0, implying that,

$$\lim_{t\to\infty}\left\{\max_{s\notin\mathcal{F}}|E(s;V_t)|\right\}=0.$$

In other words, the Bellman errors of starving states also converge to 0. Therefore, we have shown that $\lim_{t\to\infty}|E(s;V_t)|=0$ for *all* states $s\in\mathcal{S}$, and thus completed the asymptotic convergence proof since zero Bellman error implies an exact optimal value function. □

3.4.2 Proof of Theorem 1

Proof (of Theorem 1). Since $H_t(s)=|E(s;V_t)|$ for all $s\in\mathcal{S}$ and all $t\geq 1$ in GenPS, Condition 1 of Lemma 7 follows immediately from Equation 3.5, and Condition 2 is satisfied with constant $C=1$. Therefore, GenPS converges to the optimal value function, by Lemma 7. □

3.4.3 Proof of Theorem 2

Proof (of Theorem 2). We first verify that Condition 1 of Lemma 7 holds in PS. For $t>\tau$, let t' be the last time s_t is chosen for a Bellman backup. Since s_t is chosen for backups infinitely often if $t>\tau$, we can always find a t large enough so that t' exists. It follows from Equation 3.3 that state s_t's priority value at time t, $H_t(s_t)$, can be written equivalently as:

$$H_t(s_t)=\max_{t'\leq t_0<t}\left[|E(s_{t_0};V_{t_0})|\max_a[T(s_{t_0}|s_t,a)]\right].$$

Since the right-hand side above converges to 0, due to Equation 3.5, $H_t(s_t)$ also converges to 0. Thus, Condition 1 of Lemma 7 is satisfied.

Verifying Condition 2 is more involved. Let s be any state outside of $\mathcal{F}$. We first

consider the case of $s = s_t$ for some $t \leq \tau$. On one hand,

$$\begin{aligned} E(s; V_{t+1}) &= \max_{a \in \mathcal{A}} [E^a(s; V_t) + \gamma T(s_t \mid s_t, a) E(s_t; V_t)] - E(s_t; V_t) \\ &\leq \max_{a \in \mathcal{A}} [E^a(s; V_t)] + \gamma \max_{a \in \mathcal{A}} [T(s_t \mid s_t, a) E(s_t; V_t)] - E(s_t; V_t) \\ &= \gamma \max_{a \in \mathcal{A}} [T(s_t \mid s_t, a) E(s_t; V_t)] \\ &\leq \gamma \max_{a \in \mathcal{A}} [T(s_t \mid s_t, a)] \, |E(s_t; V_t)| \\ &= \gamma H_{t+1}(s_t), \end{aligned}$$

where the first equality is justified by Lemma 8, the first inequality is due to the fact that $\max_x [f(x) + g(x)] \leq \max_x f(x) + \max_x g(x)$, the second equality is due to Equation 3.8 and $s = s_t$, and the last equality is due to Equation 3.3. On the other hand, let $a_t = \arg\max_{a \in \mathcal{A}} E^a(s_t; V_t)$, then we have

$$\begin{aligned} E(s; V_{t+1}) &= \max_{a \in \mathcal{A}} [E^a(s; V_t) + \gamma T(s_t \mid s_t, a) E(s_t; V_t)] - E(s_t; V_t) \\ &\geq E^{a_t}(s; V_t) + \gamma T(s_t \mid s_t, a_t) E(s_t; V_t) - E(s_t; V_t) \\ &= \gamma T(s_t \mid s_t, a_t) E(s_t; V_t) \\ &\geq -\gamma T(s_t \mid s_t, a_t) \, |E(s_t; V_t)| \\ &\geq -\gamma \max_{a \in \mathcal{A}} [T(s_t \mid s_t, a)] \, |E(s_t; V_t)| \\ &= -\gamma H_{t+1}(s_t), \end{aligned}$$

where the first equality is again due to Lemma 8, the second equality is by definition of a_t and $s = s_t$, and the last equality is due to Equation 3.3. Therefore, we have proved for all t that $|E(s_t; V_{t+1})| \leq \gamma H_{t+1}(s_t)$, or equivalently,

$$H_{t+1}(s_t) \geq \frac{1}{\gamma} |E(s_t; V_{t+1})| . \tag{3.10}$$

In general, a similar analysis for $s \neq s_t$ seems impossible. So, we use another trick. For any state $s \notin \mathcal{F}$, let $\tau_s \leq \tau$ be the *last* time that it is chosen for update by PS.[12] Due to Equation 3.10, we have

$$H_{\tau_s+1}(s) \geq \frac{1}{\gamma} |E(s; V_{\tau_s+1})| . \tag{3.11}$$

[12]We have verified Condition 1 of Lemma 7, which requires that the priority values of non-starving states converge to 0 in the limit. Therefore, any state s will be chosen for Bellman backup at least once if $H_1(s) > 0$, since $H_t(s)$ is non-decreasing w.r.t. t if s is *never* chosen for Bellman backup. This justifies the well-definedness of τ_s.

As verified above, $H_t(s)$ converges to 0 in the limit, and it is non-decreasing for $t > \tau_s$, due to Equation 3.3. Therefore, $H_{\tau_s+1}(s) = 0$, implying $E(s; V_{\tau_s+1}) = 0$. Furthermore, we can show that $E(s; V_t) = 0$ for all $t > \tau_s$. Otherwise, let t be the first time after τ_s in which $E(s; V_t) \neq 0$. This event can happen only when the Bellman backup on s_t at step t makes the Bellman error of s be nonzero; namely, $|E(s_t; V_t)| \max_{a \in A}[T(s|s_t, a)] \neq 0$. But, it also causes $H_{t+1}(s) \neq 0$, contradicting the fact that $H_t(s) = 0$ for *all* $t > \tau_s$. Since both $H_t(s)$ and $E(s; V_t)$ are zero, Condition 2 is satisfied trivially with constant $C = 1$. □

3.4.4 Proof of Theorem 3

Two lemmas will be useful. The first says that the Bellman error in IPS is non-negative; in other words, the value function cannot decrease over time. The second says that the denominator in IPS's priority function, $V_{t+1}(s) + E(s; V_{t+1}(s))$, is never close to 0. Since we only consider Markov chains (MDPs with a single action), we suppress dependences on actions to simplify our notation.

Lemma 9 *During the execution of IPS in a Markov chain, $E(s; V_t) \geq 0$ for all t. That is, $V_t(s)$ is non-decreasing over time, since $V_{t+1}(s) = V_t(s) + \mathbb{I}(s = s_t)E(s; V_t)$.*

PROOF. We prove the lemma by mathematical induction. For $t = 1$, we have

$$\begin{aligned} E(s; V_1) &= R(s) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s) V_1(s') - V_1(s) \\ &= R(s) + \gamma \left(-\frac{1}{1-\gamma} \right) - \left(-\frac{1}{1-\gamma} \right) \\ &= R(s) + 1 \ \geq \ 0, \end{aligned}$$

since we have assumed $R(s) \geq -1$. For the inductive step, assume $E(s; V_{t-1}) \geq 0$ for some $t > 1$, and consider any state s at timestep t. If $s = s_{t-1}$, then according to Lemma 8, we have

$$\begin{aligned} E(s; V_t) &= E(s; V_{t-1}) + \gamma T(s_{t-1} \mid s_{t-1}) E(s_{t-1}; V_{t-1}) - E(s_{t-1}; V_{t-1}) \\ &= \gamma T(s_{t-1} \mid s_{t-1}) E(s_{t-1}; V_{t-1}) \geq 0. \end{aligned}$$

On the other hand, if $s \neq s_{t-1}$, Lemma 8 implies that

$$E(s; V_t) = E(s; V_{t-1}) + \gamma T(s_{t-1} \mid s) E(s_{t-1}; V_{t-1}) \geq 0.$$

So in either case, $E(s; V_t) \geq 0$ and thus the lemma is proved by mathematical induction. □

Lemma 10 *During the execution of IPS in a Markov chain,* $E(s; V_t) + V_t(s) \leq -\nu$ *for all* t *and all* $s \neq s_g$.

PROOF. We will first prove the fact that $V_t(s) \leq V^*(s)$ for all t by mathematical induction. This statement is true for $t = 1$ as the algorithm uses pessimistic initialization: $V_1(s) \leftarrow -1/(1-\gamma)$. Suppose $V_{t-1}(s) \leq V^*(s)$ for all $\tau < t$ for some $t > 1$ and consider timestep t for any state s. If $s \neq s_{t-1}$, then $V_t(s) = V_{t-1}(s) \leq V^*(s)$; otherwise,

$$V_t(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s) V_{t-1}(s') \leq R(s) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s) V^*(s') = V^*(s),$$

where the inequality makes use of the inductive hypothesis that $V_{t-1}(s') \leq V^*(s')$, and last step is due to the Bellman equation. We have thus proved $V_t(s) \leq V^*(s)$ for all t. The lemma then follows immediately from this fact, since for any $s \neq s_g$,

$$\begin{aligned} E(s; V_t) + V_t(s) &= R(s) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s) V_t(s') \\ &\leq R(s) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s) V^*(s') \\ &= V^*(s) \leq -\nu. \end{aligned}$$

□

PROOF (of Theorem 3). We verify both conditions in Lemma 7. First, note that

$$H_{t+1}(s) = -\frac{|E(s; V_{t+1})|}{V_{t+1}(s) + E(s; V_{t+1}(s))} \leq \frac{|E(s; V_{t+1})|}{\nu},$$

where the first step is due to Lemma 9 and the second to Lemma 10. According to Lemma 7, $|E(s; V_t)|$ converges to 0 in the limit. It follows that $H_t(s)$ also converges to 0 in the limit, and so Condition 1 is satisfied. Similarly, we have

$$H_{t+1}(s) = -\frac{|E(s; V_{t+1})|}{V_{t+1}(s) + E(s; V_{t+1}(s))} \geq -\frac{|E(s; V_{t+1})|}{V_1(s)} = (1-\gamma)\,|E(s; V_{t+1})|\,,$$

which verifies that Condition 2 also holds. Thus, by Lemma 7, IPS converges to the exact value function V^* in the limit. □

Chapter 4

Reinforcement Learning in Markov Decision Processes

This chapter surveys a number of reinforcement-learning algorithms, including exact ones for finite MDPs and approximate ones with function approximation. Of particular interest is the exploration/exploitation tradeoff and *ad hoc* heuristics approaches, which motivate the formal notion of PAC-MDP as a mathematical framework for evaluating the efficiency of exploration.

4.1 Reinforcement Learning in Finite Markov Decision Processes

This section surveys a small number of classical reinforcement-learning algorithms for finite MDPs. Similar to exact planning algorithms in the previous section, we assume that the value function and policies are represented by lookup tables and the goal is to learn an exact optimal policy or value function.

We start in §4.1.1 with *model-free* algorithms that directly learn optimal value functions from which optimal policies can be derived using Equation 2.7. In contrast, the *model-based* ones in §4.1.2 are indirect in the sense that they first learn the MDP model and then plan in the MDP to obtain an optimal policy. It is worth mentioning a third class of algorithms known as policy search that attempt to search for an optimal policy directly in a pre-given policy space [Baxter and Bartlett 2001; Konda and Tsitsiklis 2000; Sutton et al. 2000]. Many reinforcement-learning algorithms are not covered here, such as adaptive heuristic critic [Barto et al. 1983] and adaptive real-time dynamic programming [Barto et al. 1995].

4.1.1 Temporal Difference

Temporal difference or TD [Sutton 1988; Watkins 1989] is a family of algorithms for solving online reinforcement learning as defined in Definition 3. They can be viewed as stochastic approximation algorithms based on the Bellman equation (Equation 2.12). When the environment is an ergodic Markov chain, the TD(λ) algorithm of Sutton [1988] can be used to learn the value function exactly in the limit. Here, we focus on another algorithm known as Q-learning [Watkins 1989] that aims at learning the optimal value function of a finite MDP, and is similar to an algorithm called Sarsa [Sutton 1996].

Algorithm 7 gives a detailed description of Q-learning. The algorithm requires a sequence of learning rates, $[\alpha_t]_{t\in\mathbb{N}}$, where $\alpha_t \in (0,1)$. It is known that [Watkins and Dayan 1992; Tsitsiklis 1994] the value function in Q-learning converges to Q^* asymptotically with probability 1 as long as the following conditions hold for every state–action pair, (s,a):

$$\sum_{i=1}^{\infty} \alpha_{t_i} = \infty \qquad \text{and} \qquad \sum_{i=1}^{\infty} \alpha_{t_i}^2 < \infty, \tag{4.1}$$

where $t_1, t_2, \ldots$ are the timesteps in which action a is taken in state s (that is, $s_{t_i} = s$ and $a_{t_i} = a$). These two conditions are often needed in proving asymptotic convergence of stochastic approximation algorithms [Robbins and Monro 1951]. Intuitively, the first condition guarantees that the sequence of learning rates is sufficiently large to move the initial $Q(s,a)$ value to approach $Q^*(s,a)$, while the second guarantees that the updates on $Q(s,a)$ decreases fast enough so that $Q(s,a)$ eventually stablizes at $Q^*(s,a)$. Similar convergence guarantees are established for variants of Q-learning, including TD(0) and Sarsa [Sutton 1988; Szepesvári and Littman 1999; Singh et al. 2000].

Since the learning rate α_t is less than 1, the first condition in Equation 4.1 implies that every action a must be tried in every state s infinitely often. Obviously, this requirement is unavoidable for convergence of Q-learning unless further assumptions are made: if the agent wants to learn $Q^*(s,a)$ exactly in a stochastic MDP, it has to apply action a infinitely many times in state s to recover $R(s,a)$ and $T(\cdot \mid s,a)$ precisely. This fact is related to the exploration/exploitation dilemma that we describe in §1.2 and elaborate more in §4.3. Among many natural exploration strategies [Thrun 1992], the

"ϵ-greedy" rule is popular in reinforcement learning. An ϵ-greedy agent simply takes a greedy action with respect to its current value-function estimate with probability $1-\epsilon$ and a random action with probability ϵ. If an MDP is ergodic, then every action will be tried in every state infinitely often, and thus Q-learning converges to Q^* asymptotically with probability 1.

Algorithm 7 Q-learning.

0: **Inputs:** $\mathcal{S}, \mathcal{A}, \gamma$.
1: Initialize $Q(s,a)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$
2: **for all** $t = 1, 2, \ldots$ **do**
3: Observe current state s_t, take action a_t, observe reward r_t, and transition to a new state s_{t+1}.
4: Update Q-function by

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

5: **end for**

4.1.2 Certainty Equivalence

Q-learning in the previous subsection works by learning the optimal value function based on online transitions without estimating rewards and transitions of the underlying MDP. It is thus called *model free*. In contrast, *model-based* approaches are more straightforward: they often learn the reward and transition functions of the MDP either implicitly or explicitly, and then compute the optimal policy in the learned MDP using any of the MDP planning algorithms surveyed in §3.1. This subsection describes certainty equivalence [Kumar and Varaiya 1986], a representative model-based algorithm.

At time t, let the MDP estimate be $\hat{M}_t = \langle \mathcal{S}, \mathcal{A}, \hat{T}_t, \hat{R}_t, \gamma \rangle$, where $\hat{T}_t$ and $\hat{R}_t$ are maximum-likelihood estimates of the true transition and reward functions, respectively. Formally,

$$\hat{T}_t(s' \mid s, a) \stackrel{\text{def}}{=} \frac{N_{s,a,s'}(t)}{N_{s,a}(t)} = \frac{\sum_{\tau=1}^{t-1} \mathbb{I}(s_\tau = s, a_\tau = a, s_{\tau+1} = s')}{\sum_{\tau=1}^{t-1} \mathbb{I}(s_\tau = s, a_\tau = a)} \tag{4.2}$$

$$\hat{R}_t(s,a) \stackrel{\text{def}}{=} \frac{1}{N_{s,a}(t)} \sum_{\tau=1}^{t-1} r_\tau \mathbb{I}(s_\tau = s, a_\tau = a), \tag{4.3}$$

where $N_{s,a}(t)$ is the number of times action a is taken in state s before timestep t, and

$N_{s,a,s'}(t)$ is the number of times state s' is reached after the agent takes action a in state s before timestep t. When $N_{s,a}(t) = 0$, various adjustments to Equations 4.2 and 4.3 are possible, such as adding 1 to the denominator.

An implementation of certainty equivalence is given in Algorithm 8, where the MDP estimate is updated and a new policy is re-computed at every step. This choice is computationally expensive but simplifies the exposition. It is expected that smarter, but more complicated, variants of Algorithm 8 are more efficient in practice.

Algorithm 8 A version of certainty equivalence.

0: **Inputs:** $\mathcal{S}, \mathcal{A}, \gamma$.
1: **for all** $t = 1, 2, \ldots$ **do**
2: Compute an MDP estimate, $\hat{M}_t$, using Equations 4.2 and 4.3.
3: Compute an optimal policy, π_t^*, of the MDP $\hat{M}_t$.
4: Observe current state s_t, take action $a_t = \pi_t^*(s_t)$ (possibly with some exploration rule such as ϵ-greedy), observe reward r_t, and transition to a next state s_{t+1}.
5: **end for**

The simulation lemma (*c.f.*, Lemma 33) implies that if the true MDP model parameters are learned to within sufficient accuracy, then the optimal policy in the learned MDP is near-optimal in the true MDP. Therefore, if certainty equivalence is able to collect data so that $N_{s,a}(t) \to \infty$ in the limit for all (s, a), then the estimated model converges to the true model, and thus the policy π_t^* converges to π^*.

4.1.3 Dyna

Certainty equivalence and Q-learning represent two extremes: the former performs a complete planning step when it updates its policy based on the estimate MDP, while the latter performs extremely simple planning by one-step lookahead. Although certainty equivalence seems to make the best use of data, it is computationally demanding. In contrast, Q-learning is computationally cheap, but makes inefficient use of data as every transition is used once. Middle-ground solutions seek to combine the best of both worlds. A representative solution is Dyna [Sutton 1990; 1991].

Dyna maintains an estimated MDP like certainty equivalence, and also acts based on an estimate of the Q-function like Q-learning. Instead of a complete replanning in every step like certainty equivalence, Dyna performs *shallow* planning. This computation can

be achieved in different ways, such as:

- select a subset of previous transitions and perform Q-learning-like backups (*e.g.*, the experience-replay technique of Lin [1992]);
- select a subset of states, for each s of them pick an action a, *simulate* a transition from (s, a) using the model, and then do Q-learning-like backups (*e.g.*, the Dyna-Q algorithm of Sutton [1990]); and
- select a subset of states and perform Bellman backups in those states using the estimated transition and reward functions (*e.g.*, the PS algorithm of Moore and Atkeson [1993]; *c.f.*, §3.1.2).

Dyna was first proposed [Sutton 1990; 1991] with the subset of states being chosen randomly for shallow planning. It was then shown independently by Moore and Atkeson [1993] and Peng and Williams [1993] that it is beneficial to select such states in an informed manner.

4.2 Approximate Reinforcement Learning

As we shall see, approximate reinforcement learning bears many similarities to approximate planning. A fundamental reason is that many approximate planning algorithms avoid computational complexity that is explicitly dependent on the size of the MDP by using samples of transitions of the MDP. For instance, one of the algorithms we will discuss is based on approximate policy iteration. When the learning algorithm is similar to the planning algorithm, we will mention it briefly without going into details.

Since the motivation of studying approximate RL is to solve large-scale MDPs, we will use compact function representations such as those covered in §3.2.

4.2.1 State Abstraction

Similar to discretization/abstraction-based planning algorithms, an approximate RL algorithm may discretize the state/action space of a possibly continuous MDP to obtain a smaller, finite MDP, and then run any of the algorithms for finite MDPs in §4.1. Adaptive discretization such as the U-tree algorithm [McCallum 1995] adaptively refines

the piecewise constant value function estimate (which is represented as a regression tree) based on observed state transitions. See Moore and Atkeson [1995] and Bernstein and Shimkin [2008] for more examples.

4.2.2 Incremental Temporal Difference

The temporal-different methods in §4.1.1 can be modified naturally when the value function is represented compactly by: $Q(s,a) = Q_{\mathbf{w}}(s,a)$, where $Q_{\mathbf{w}}(\cdot,\cdot)$ is a parametric function with parameter vector $\mathbf{w} \in \mathbb{R}^k$ for some $k \in \mathbb{N}$. For instance, if state discretization is used, then $Q_{\mathbf{w}}$ is a piecewise constant function, and components of $\mathbf{w}$ correspond to the function values in the discretized states. As another example, if $Q_{\mathbf{w}}$ is linearly parameterized, then $\mathbf{w}$ is the weight vector defining the linear function. As a third example, if $Q_{\mathbf{w}}$ is represented by a neural network with fixed topology, then $\mathbf{w}$ consists of weights associated with links in the network.

Q-learning for finite MDPs (Algorithm 7) can be modified to be Algorithm 9 when function approximation is used, with only two changes in the initialization step and update rule (Lines 1 and 4, respectively). Here, $\mathbf{w}$ may be initialized in different ways. Learning occurs on the parameter vector instead of the Q-values directly. An important special case is when Q is linear in some pre-defined basis function, $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$,

$$Q_{\mathbf{w}}(s,a) \stackrel{\text{def}}{=} \mathbf{w}^\top \boldsymbol{\phi}(s,a),$$

in which case the update rule in Line 4 of Algorithm 9 becomes

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_t \left(r_t + \gamma \max_{a \in \mathcal{A}} \mathbf{w}_t^\top \boldsymbol{\phi}(s_{t+1}, a) - \mathbf{w}_t^\top \boldsymbol{\phi}(s_t, a_t) \right) \boldsymbol{\phi}(s_t, a_t).$$

Extensions similar to Line 4 of Algorithm 9 can be applied to variants of Q-learning, such as TD(λ) [Sutton 1988] and Sarsa [Sutton 1996]. Unfortunately, these stochastic-approximation-style updates do not necessarily guarantee convergence. Mixed results are known such as the tremendous empirical success of applying neural network as the function approximator in Backgammon [Tesauro 1995] and the unfortunate divergence results reported in the literature [Boyan and Moore 1995; Baird 1995]. Special cases are investigated to establish convergence guarantees. Such cases often rely on the

Algorithm 9 Q-learning with function approximation.

0: **Inputs:** $\mathcal{S}, \mathcal{A}, \gamma$.
1: Initialize $\mathbf{w}$.
2: **for all** $t = 1, 2, \ldots$ **do**
3: Observe current state s_t, take action a_t, observe reward r_t, and transition to a new state s_{t+1}.
4: Update parameter by

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t \left(r_t + \gamma \max_{a \in \mathcal{A}} Q_{\mathbf{w}}(s_{t+1}, a) - Q_{\mathbf{w}}(s_t, a_t) \right) \nabla_{\mathbf{w}} Q_{\mathbf{w}}(s_t, a_t).$$

5: **end for**

fact that the Bellman operator when combined with special function approximators is a contraction [Tsitsiklis and Van Roy 1997; Szepesvári and Smart 2004; Melo and Ribeiro 2007; Melo et al. 2008]; see §3.3.2 for more discussions.

Another approach known as residual gradient is due to Baird [1995]. It changes the update rule of Algorithm 9 to perform stochastic gradient descent on the squared Bellman error. Although the algorithm is limited by a double-sample requirement to guarantee unbiasedness, its value function cannot diverge. However, this algorithm can be slow in learning [Baird 1995; Schoknecht and Merke 2003] or suffer suboptimal online prediction performance [Li 2008].

A third approach known as grow-support is proposed by Boyan and Moore [1995], which assumes the optimal value function can be represented exactly by the approximation scheme. The algorithm maintains a "support set" of states whose optimal values have been computed sufficiently accurately, and is initialized to the empty set at the beginning. When sample transitions accumulate, the support set grows and eventually the optimal value function is computed for all states.

4.2.3 Approximate Policy Iteration

Approximate policy iteration (Algorithm 6) can be used to solve reinforcement-learning problems as well. Different ways to do the approximate policy-evaluation and policy-improvement steps result in different API-based algorithms. For instance, when rollouts are used to estimate $\hat{Q}^{\pi_t}$ and a classifier mapping states to the greedy action with respect

to $\hat{Q}^{\pi_t}$ are used, we have a classifier-based RL algorithm [Lagoudakis and Parr 2003b; Li 2004; Li et al. 2007]. Other examples include an algorithm that uses kernel regression for policy evaluation [Ormoneit and Sen 2002]. In this section, we focus on least-squares policy iteration or LSPI [Lagoudakis and Parr 2003a], an algorithm that employs least-squares techniques for policy evaluation based on linear function approximation. A more general algorithm is studied by Antos et al. [2008].

In its original form, LSPI is an *offline* algorithm. Instead of requiring data to be obtained from online interaction with the environment (Definition 1), the algorithm assumes that a set of sampled transitions are provided without requirements on how they are collected. Precisely, the sample set $\mathcal{D}$ contains m data points:

$$\mathcal{D} = \{(s_i, a_i, r_i, s'_i) \mid i = 1, 2, \ldots, m\},$$

where the i-th sample, (s_i, a_i, r_i, s'_i), provides a piece of information for the dynamics in (s_i, a_i): the agent takes action a_i in state s_i, receives an immediate reward r_i and reaches a next state s'_i. Algorithm 10 gives a pseudo-code description of the algorithm. In practice, various approaches can be used to decide when to terminate the algorithm [Lagoudakis and Parr 2003a]. Furthermore, robust matrix inversion like pseudo-inverse [Golub and Van Loan 1996] is needed when the matrix $\hat{A}$ is ill-conditioned.

Algorithm 10 Least-squares policy iteration.

Inputs: $\mathcal{S}, \mathcal{A}, \gamma, \mathcal{D} = \{(s_i, a_i, r_i, s'_i) \mid i = 1, 2, \ldots, m\}, \boldsymbol{\phi} \in \left(\mathbb{R}^k\right)^{\mathcal{S}\times\mathcal{A}}$
Initialize $\mathbf{w}_1 \in \mathbb{R}^k$ arbitrarily.
for $t = 1, 2, 3, \ldots$ **do**
  $\hat{A} \leftarrow O_{k\times k}$ {the $k \times k$ zero matrix}
  $\hat{b} \leftarrow \mathbf{0}_k$ {the zero vector of dimension k}
  for $i = 1, 2, \ldots, m$ **do**
    $a'_i \leftarrow \operatorname{argmax}_{a\in\mathcal{A}} Q_{\mathbf{w}_t}(s'_i, a) = \operatorname{argmax}_{a\in\mathcal{A}} \mathbf{w}_t^\top \boldsymbol{\phi}(s'_i, a)$.
    $\hat{A} \leftarrow \hat{A} + \boldsymbol{\phi}(s_i, a_i)\left(\boldsymbol{\phi}(s_i, a_i) - \gamma\boldsymbol{\phi}(s'_i, a'_i)\right)^\top$.
    $\hat{b} \leftarrow \hat{b} + \boldsymbol{\phi}(s_i, a_i) r_i$.
  end for
  $\mathbf{w}_{t+1} \leftarrow \hat{A}^{-1}\hat{b}$.
end for

Like generic API algorithms, LSPI has the policy-evaluation and policy-improvement steps in every iteration. The intermediate policies are represented *implicitly* as the greedy policy with respect to the linear value functions. In iteration t, the weight

vector is $\mathbf{w}_t$, and the policy is

$$\pi_t(s) \stackrel{\text{def}}{=} \operatorname*{argmax}_{a \in \mathcal{A}} \mathbf{w}_t^\top \boldsymbol{\phi}(s, a).$$

The key component of LSPI is its policy-evaluation step (Lines 3—10 in Algorithm 10), which is called LSTDQ [Lagoudakis and Parr 2003a]. This algorithm can be viewed as an off-line, off-policy version of the least-squares temporal difference (LSTD) algorithm [Bradtke and Barto 1996] applied to an induced Markov chain (*c.f.*, §2.1) whose state space is $\mathcal{S} \times \mathcal{A}$. A useful extension known as LSTD(λ) is due to Boyan [2002] who showed the potential advantage of using a nonzero λ value. LSTD is identical to LSTD(0).

Assuming $\mathcal{A}$ is finite and it takes $\mathcal{O}(k)$ time to compute the basis $\boldsymbol{\phi}$, the computational complexity of LSTDQ in Algorithm 10 is $\mathcal{O}(km\,|\mathcal{A}| + k^2m + k^3)$: the first term is from the computation of greedy actions in state s'_i, the second from the construction of matrix $\hat{A}$ and vector $\hat{b}$, and the last from the matrix inversion in Line 10. Although the linear dependence on m and $|\mathcal{A}|$ is reasonable, the cubic dependence on k makes the algorithm intractable except for small value of k. Recently, recursive least-squares techniques were applied to LSTD and LSTDQ, so that the matrix inverse, $\hat{A}^{-1}$, is computed iteratively using the Sherman-Morrison formula [Golub and Van Loan 1996], resulting in an $\mathcal{O}(k^2)$ complexity [Xu et al. 2002]. More recently, techniques for solving sparse linear systems motivate even more efficient LSTD-like algorithms that enjoys $\mathcal{O}(k)$ complexity [Geramifard et al. 2006; 2007; Yao and Liu 2008]. Interested readers are referred to the discussions at the end of §3.2.2 for related papers on feature selection in reinforcement learning when linear value-function approximation is used.

Motivated by successes of the kernel trick in machine learning [Shawe-Taylor and Cristianini 2004], Xu et al. [2005] introduced the kernel LSTD(λ) algorithm, the use of which in the general API framework results in kernel LSPI [Xu et al. 2007]. Recently, Taylor and Parr [2009] gave an analysis of Bellman errors when kernel methods are used in temporal difference learning.

4.2.4 Other Algorithms

There are many other RL algorithms that cannot be covered in detail in this dissertation. For example, policy-search and policy-gradient algorithms search a pre-defined policy space for a near-optimal policy [Sutton et al. 2000; Baxter and Bartlett 2001; Ng and Jordan 2000]; Actor-critic algorithms [Barto et al. 1983; Sutton 1984; Konda and Tsitsiklis 2003; Peters et al. 2005] consist of two interacting components: The *actor* maintains and updates a policy for choosing actions, while the *critic* learns a value function for that policy incrementally. Motivated by the success of Dyna in finite MDPs, Sutton et al. [2008] introduce the use of linear value functions to Dyna in continuous MDPs. Finally, we note that the RL problem can be formulated as an optimization problem and can be approached by many general-purpose algorithms such evolutionary algorithms [Moriarty et al. 1999; Whiteson and Stone 2006].

4.3 Exploration in Reinforcement Learning

This section reviews previous work on balancing exploration and exploitation. We start with the K-armed bandit problems, which can be viewed as a special case of MDPs, and then discuss Bayesian exploration, which is optimal in a sense but computationally expensive. Finally, *ad hoc* heuristics are described with examples showing why they may fail. These negative results motivate a formal framework to evaluate efficiency of exploration in §4.4.3.

4.3.1 Exploration in K-Armed Bandit Problems

Many of the exploration strategies for MDPs find their analogues in a specific type of MDP called *K-armed bandits* [Berry and Fristedt 1985]. It suffices to focus on this problem for the purpose of the dissertation. Interested readers are referred to the literature on several variants of it, such as adversarial bandits [Auer et al. 2002b; Auer 2002], non-stationary bandits [Auer et al. 2002b; Auer 2002], associative bandits [Kaelbling 1994; Auer 2002; Abe et al. 2003; Strehl et al. 2006d], and budgeted bandits [Madani et al. 2004]. Although the terms "arm" and "payoff" are typically used in the K-armed

bandit literature, we will instead use "action" and "reward" to be consistent with the terminology in the rest of the dissertation.

A K-armed bandit is an MDP M with a single state (so $|\mathcal{S}| = 1$) and K actions (so $|\mathcal{A}| = K$). Rewards due to taking an action are stochastic as in general MDPs, but the transition always falls back to the same state. Since there is only one state, we suppress s and denote the reward function by $R(a)$. The optimal policy in a K-armed bandit is to choose the action with maximum expected reward:

$$a^* = \operatorname*{argmax}_{a \in \mathcal{A}} R(a).$$

Without knowledge of the function $R(\cdot)$, the agent has to explore different actions and hopefully converges to the optimal action eventually. To evaluate how fast the agent learns in this process, two natural formalisms can be used.

Regret Analysis

The *expected regret* is the expected difference between cumulative expected rewards of the optimal action and the actions chosen by the agent. Precisely, let a_t be the action chosen at timestep t, then the expected regret up to time $H \in \mathbb{N}$ is defined by

$$L(H) \stackrel{\text{def}}{=} H \cdot R(a^*) - \sum_{t=1}^{H} R(a_t).$$

By definition, it is clear that $L(H) \geq 0$ for any H. If $L(H) = o(H)$, it implies the agent will choose the optimal action almost always in the limit. The smaller $L(H)$ is, the faster the agent is expected to converge to (near-)optimal actions.

A regret lower bound of $\Omega(\ln H)$ in K-armed bandits has been known for a long time. Lai and Robbins [1985] show an algorithm that achieves this lower bound asymptotically. Later, more algorithms with $\Theta(\ln H)$ regret bound were developed.

More recently, Auer et al. [2002a] proposed an algorithm called UCB (standing for "upper confidence bound") that achieves a $\Theta(\ln H)$ regret *uniformly* over time. The algorithm is simple and sheds lights on some efficient exploration methods for MDPs. For time $t \in \mathbb{N}$ and action $a \in \mathcal{A}$, denote by (similar to Equation 4.3)

$$\hat{R}_t(a) = \frac{1}{N_a(t)} \sum_{\tau=1}^{t-1} r_\tau \mathbb{I}(a_\tau = a)$$

the maximum-likelihood estimate of the average reward of action a at time t, where $N_a(t)$ is the number of times action a has been chosen before timestep t. A purely exploiting agent will choose an action that maximizes the empirical average reward so far; that is,

$$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} \hat{R}_t(a).$$

But since these finite-sample estimates of average rewards are imprecise, the action with the highest empirical average reward may not be the optimal action a^*. Thus, it may be beneficial in the long run to try alternative actions in case we have underestimated their rewards. Naturally, we have a higher confidence in our maximum-likelihood estimate when it is computed using more data. If few data are available for estimating the reward of an action, we would like to encourage the agent to explore this action to refine the estimate of its reward. The UCB algorithm uses exactly this idea by applying:

$$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ \hat{R}_t(a) + \sqrt{\frac{2 \ln(t-1)}{N_a(t)}} \right\}.$$

The second term above is a confidence interval, and thus the sum is an upper confidence bound of the true reward, $R(a)$. The confidence-interval term is an exploration bonus that encourages trying infrequently chosen actions. Auer [2002] also show that a similar idea can be used in variations of K-armed bandit problems, such as in the adversarial setting. Finally, the idea of UCB has been used in sampling-based approximate planning for large-scale MDPs (§3.3.4).

PAC Analysis

Another way to quantify learning speed in K-armed bandits is through sample complexity in the *Probably Approximately Correct (PAC)* framework [Valiant 1984]. Although *PAC* was originally proposed for supervised learning (*c.f.*, §5.4), it can be defined similarly for studying bandit problems. Roughly speaking, the PAC sample complexity is an upper bound of the number of timesteps a learner needs to identify, with high probability, a near-optimal action of the bandit.

Formally, let $\mathbf{A}$ be a K-armed bandit learning algorithm. The algorithm is allowed to try the K actions in a sequential order, and then must terminate with an output

$\hat{a}^* \in \mathcal{A}$ after finitely many timesteps. The algorithm is also allowed to be randomized. Due to randomness in the algorithm and in the rewards observed, both the number T of timesteps $\mathbf{A}$ takes to terminate and its output action $\hat{a}^*$ are stochastic.

Definition 6 *Let* $\mathbf{A}$ *be a* K*-armed bandit learning algorithm that terminates after* T *timesteps with action* $\hat{a}^*$*. A function* $\zeta(\epsilon, \delta)$ *is called a* sample complexity *of* $\mathbf{A}$*, if for any* $\epsilon, \delta \in (0, 1)$*, the following holds with probability at least* $1 - \delta$*: (i)* $T \leq \zeta(\epsilon, \delta)$*, and (ii)* $R(a^*) - R(\hat{a}^*) < \epsilon$*.*

Matching upper and lower bounds are found for K-armed bandit problems. Even-Dar et al. [2002] describe the median elimination algorithm that eliminates actions in rounds. In each round, each surviving action is tried for some polynomial number of times, and the bottom half actions (in terms of the rewards they collect in that round) are eliminated. The algorithm continues until all but one action are eliminated, and returns this surviving action. The sample complexity of this algorithm is

$$\zeta(\epsilon, \delta) = \mathcal{O}\left(\frac{K}{\epsilon^2} \ln \frac{1}{\delta}\right).$$

Later, Mannor and Tsitsiklis [2004] show a matching lower bound. We cite their result here, which will be useful in §6 and §8:

Lemma 11 *[Mannor and Tsitsiklis 2004, Theorem 1] The sample complexity* ζ *of any* K*-armed bandit algorithm* $\mathbf{A}$ *is, in the worst case,*

$$\zeta(\epsilon, \delta) = \Omega\left(\frac{K}{\epsilon^2} \ln \frac{1}{\delta}\right).$$

4.3.2 Bayesian Exploration

We now turn to the general MDP setting that features the nature of sequential decision making; that is, actions can affect future states and thus future rewards. This is different from bandit problems where earlier actions does not affect later consequences. It turns out that this difference makes MDPs a much harder problem than bandits when studying the exploration-exploitation tradeoff.

About half a century ago, researchers started studying Bayesian approaches to exploration; see Duff [2002] for an interesting historical review. Starting with a prior distribution of parameters in the underlying MDP, these methods repeatedly update the posterior distribution of the parameters conditional on the observation history. At each time step, the action that maximizes expected future rewards with respect to the posterior distribution is taken.

Let $\mathcal{M}$ be the class of MDPs over which we have a prior distribution, p_0, and p_t be the posterior distribution over $\mathcal{M}$ after t transitions have been observed. Assume, for simplicity, that the reward function is known and the only uncertainty about the MDP is in its transition function. Then, Bayes rule implies the following formula for computing p_t based on p_{t-1} and the transition at timestep t: for any $M \in \mathcal{M}$

$$p_t(M) = \frac{T_M(s_{t+1} \mid s_t, a_t) p_{t-1}(M)}{\sum_{M' \in \mathcal{M}} T_{M'}(s_{t+1} \mid s_t, a_t) p_{t-1}(M')},$$

where T_M is the transition function of MDP model M.

Bayesian action selection is optimal in the sense that it takes the posterior into account and finds the Bayesian-optimal balance between exploration and exploitation. However, their use in practice is limited for two reasons. First, they all require a prior distribution over candidate MDPs, which is unclear in many real-world applications. Second, computing the optimal value function over posterior distributions can be a computationally extremely expensive procedure, although progress has been made recently [Duff 2002; Poupart et al. 2006]. Furthermore, we will show an example in §4.4.4 where an agent that uses Bayesian exploration fails to find the optimal policy in the limit. For these reasons, we focus on non-Bayesian and computationally tractable approaches in the dissertation.

4.3.3 Exploration Heuristics

Given the expensive operations of Bayesian exploration strategies, people have used many practical exploration heuristics; see the survey by Thrun [1992]. These heuristics work quite well in some problems, but all require trial-and-error tuning of parameters. In some problems, these *ad hoc* approaches may fail or be inefficient.

The simplest and most popular exploration rule is probably ϵ-*greedy*: the agent chooses the greedy action with respect to its current value-function estimate with probability $1-\epsilon$, and a random action with probability ϵ. If there are finitely many actions, this exploration rule can be formally defined by (assuming the greedy action is unique, for simplicity):

$$\pi(a \mid s) = \begin{cases} 1 - \frac{|\mathcal{A}|-1}{|\mathcal{A}|}\epsilon, & \text{if } a = \operatorname{argmax}_{a' \in \mathcal{A}} Q(s, a') \\ \frac{\epsilon}{|\mathcal{A}|}, & \text{otherwise} \end{cases}.$$

Clearly, 0-greedy is a purely greedy policy, while 1-greedy is a uniformly random policy that assigns equal probability mass to all actions. In practice, it is helpful to use large ϵ at the beginning and then decrease it to zero, since after a long run the agent has hopefully converged to a good enough policy and thus the fraction of exploration can be reduced.

While ϵ-greedy differentiates greedy actions from non-greedy actions, the *Boltzmann* exploration rule takes the Q-function estimates into account and embodies a soft version of ϵ-greedy. This strategy requires a *temperature parameter* $\tau \in \mathbb{R}_+$, and the action-selection probabilities are given by the following rule:[1]

$$\pi(a \mid s) = \frac{\exp\left(\frac{Q(s,a)}{\tau}\right)}{\sum_{a' \in \mathcal{A}} \exp\left(\frac{Q(s,a')}{\tau}\right)},$$

where Q is an estimated Q-function. Therefore, instead of having a hard separation between greedy and non-greedy actions, this exploration smooths the probability by taking into account the Q-values. The parameter τ controls the degree of exploration: when it is smaller, more probability mass is placed on the greedy action; in the limit of $\tau \to 0$, this exploration rule converges to a greedy policy; in the limit of $\tau \to \infty$, the rule converges to a uniformly random policy. Because Boltzmann exploration approaches the greedy action selection in the limit and the difference is governed by the temperature, this exploration rule is sometimes called *softmax*. In practice, we may start with a relatively large temperature, and then decay it as learning proceeds.

[1] It is easily verified that $\sum_{a \in \mathcal{A}} \pi(a \mid s) = 1$ for all $s \in \mathcal{S}$.

An alternative approach, called *counter-based* exploration, has shown empirical advantages over ϵ-greedy in some problems [Thrun 1992]. It requires a threshold $m \in \mathbb{N}$, and a counter c is maintained for each (s, a) pair that remembers how many times action a has been taken in state s. When the agent is in state s, it randomly picks action a such that $c(s, a) < m$; if no such action exists, a greedy action is chosen. The threshold controls the exploration-exploitation tradeoff: when m is small, the agent tries to exploit sooner, but risks at a higher probability of ending up with suboptimal policies; when m gets larger, the agent becomes more conservative and tends to explore more before exploiting and hence it is more likely to learn a near-optimal policy. While this strategy is able to explore various actions when visiting a certain state, it is myopic as it only reasons about uncertainty of actions in the current state. A similar algorithm called Rmax remedies this problem, and is discussed in more details in §7.

Another approach is to add an *exploration bonus* to the value-function estimate and then act greedily or ϵ-greedily. The bonus term decreases over time, and is somewhat similar in form to the UCB algorithm described in §4.3.1. This additive bonus has the effect of encouraging taking actions that have not been taken frequently, and may bring benefits even in non-stationary environments [Sutton 1990].

Finally, we mention another common practical choice known as *optimistic initialization* [Sutton and Barto 1998]. The idea is to initialize the value function so that $Q_1(s, a) \geq Q^*(s, a)$ for all $s \in S$ and $a \in A$. The intuition is that if an action's Q-value is optimistic in the current state, the agent is encouraged to execute that action and thus learn much of it. When sufficient learning has been completed and all Q-values are close to their true Q^*-values, selecting the maximum will guarantee near-optimal behavior. This trick can also be combined with other exploration techniques. For example, the agent can initialize its value function optimistically, and then follow the ϵ-greedy policy.

Successes have been observed for the *ad hoc* exploration rules above [Thrun 1992; Sutton and Barto 1998]. However, they may not always work effectively in all MDPs. For example, Whitehead [1991] gives an example where ϵ-greedy rule takes $\mathcal{O}(2^{|\mathcal{S}|})$ many steps even to reach the goal state for the first time, although only $\mathcal{O}(|\mathcal{S}|^2)$ many

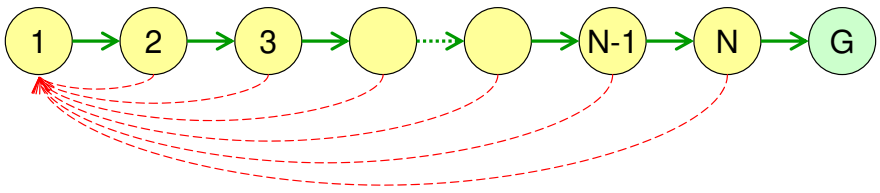


Figure 4.1: Combination lock.

steps are needed by better exploration strategies. Similarly, several common algorithms such as Q-learning with ϵ-greedy exploration and/or optimistic initialization are shown to be inefficient [Strehl 2007b, Section 4.1].

Example 8 *[Whitehead 1991] Figure 4.1 shows a combination-lock example where the ϵ-greedy rule may be inefficient. We have an MDP with $N+1$ states and 2 actions. State 1 is the start state and state G is the absorbing goal state. Taking the solid action transports the agent to the state to the right, while taking the dashed action resets the agent to the start state 1 and it has to re-start from 1 trying to get to the goal state. To simplify exposition, we use $\gamma = 1$, and $R(s,a) = -1$ for all state–actions unless $s = G$, in which case $R(s,a) = 0$.*

An agent that uses ϵ-greedy exploration rule will, at each timestep, be reset to the start state with probability at least $\epsilon/2$. Therefore, the probability that the agent always chooses the solid action until it reaches the goal is $(1-\epsilon/2)^N$, which implies the value of state N is exponentially small in N. In contrast, the optimal policy requires only N steps. This example shows that a poor exploration rule may be exponentially inefficient.

4.4 Performance Measures of Reinforcement-Learning Algorithms

Much early theoretical investigation of reinforcement-learning algorithms was on whether the algorithms converge to the optimal solution in the limit. While such theory is important for showing an algorithm is *sound* or *consistent*, it largely ignores the issue of *efficiency*. The negative (inefficiency) results mentioned in the previous section motivate a rigorous study of performance of reinforcement-learning algorithms. The next three subsections discuss three considerations that are essential to measure

the goodness of an RL algorithm: computation, space, and learning speed. Finally, we briefly mention a few other dimensions that we do not focus on in this dissertation.

4.4.1 Computational Complexity

The *per-timestep computational complexity* (or *computational complexity* for short) measures the amount of computation the algorithm has to carry out per timestep in the worst case. We do not consider the total amount of computation during the execution of an algorithm mainly because we allow the algorithm to run forever, in which case the total computation is trivially infinity. However, we note that it may be useful to consider computational complexity in the *amortized* sense [Cormen et al. 2001].

The per-timestep computation of an RL algorithm often comes from two sources: the computation required to choose actions, and the computation requires to learn (*i.e.*, to update the internal data structures and quantities like the value functions and policies). The latter depends critically on algorithms, and so only the former is discussed at a high level here.

Two operations are common in choosing actions: one is to compute a value-function estimate, and the other is to find the greedy action in a state with respect to the value-function estimate. The first operation is frequently used in model-based algorithms (*c.f.*, §4.1.2) when they re-plan by solving the optimal value function for the learned MDP model. This process can be quite expensive (see Littman [1996] for a survey). For instance, value iteration takes

$$\tilde{\mathcal{O}}\left(\frac{|\mathcal{S}|^2\,|\mathcal{A}|^2}{1-\gamma}\right)$$

time in the worst case. In contrast, model-free algorithms like Q-learning do not require this planning step but uses their value-function estimates for action selection.

The second operation—finding a (near-)greedy action—is a necessary operation for all meaningful reinforcement-learning algorithms. If the Q-function is represented by a heap [Cormen et al. 2001], finding the greedy action takes $\mathcal{O}(\ln|\mathcal{A}|)$ time. In general, $\mathcal{O}(|\mathcal{A}|)$ complexity is needed.

4.4.2 Space Complexity

The *space complexity* measures the amount of space the algorithm needs in the worst case. Q-learning, for example, requires $|\mathcal{S}|\,|\mathcal{A}|$ space to store the Q-function in a tabular manner. For incremental algorithms with linear value-function representations like Algorithm 9, the space complexity is $\mathcal{O}(k)$, the number of free parameters to learn. LSPI, however, requires $\mathcal{O}(k^2)$ space to store the matrix $\hat{A}$ as well as additional $\mathcal{O}(m)$ space to store the sample set $\mathcal{D}$.

4.4.3 Some Notions of Learning Complexity

As opposed to computational and space complexities that measures how feasible an algorithm is in terms of computing *resources*, the third quantity, *learning complexity*, measures how much *information* is needed by the algorithm to learn to achieve near-optimal policies. However, such a notion is not easy to define as in other learning problems like supervised learning [Kearns and Vazirani 1994]. A fundamental challenge in reinforcement learning is the interactive nature in the sequential-decision-making scenario. In this subsection, we examine natural extensions of a few notions of learning complexity in supervised learning, and discuss their limitations. We then, in the following subsection, define the *sample complexity of exploration*, which addresses some of the limits and will be the focus of the dissertation.

The first such choice is *regret*, an idea borrowed from competitive analysis of bandit algorithms (§4.3.1). The idea is to compare the performance of an algorithm, where "performance" is problem dependent such as prediction mistakes, to the best possible in hindsight. In the case of reinforcement learning, we may also analyze the regret, which is difference between the largest total reward obtained by the best policy and that of the algorithm [Auer and Ortner 2007]. In a sense, the regret is exactly what a reward-maximizing algorithm tries to minimize. However, analysis of regret is significantly harder and requires rather restricted assumptions that are not needed in our sample-complexity analysis. Furthermore, the regret bounds obtained are often weaker than the sample-complexity bounds we present in this dissertation.

If the algorithm has some prior information about the true, underlying MDP, it can adopt the Bayesian approach (§4.3.2): at every timestep, it maintains a posterior distribution of the underlying MDP based on the prior and the data it has observed so far, and computes the Bayesian optimal action that maximizes the expected future return. This class of methods, although optimal in the Bayesian sense, are computationally intractable in general, and as we will show soon, do not necessarily converge to a near-optimal value function or policy. Furthermore, specifying the prior distribution over MDPs may not be easy in many applications, and an unfortunately specified prior may prevent convergence to the true MDP model even if infinitely many data are given [Diaconis and Freedman 1986].

A third approach is to study asymptotic convergence rate of the value function to the optimal value function. Due to the need for exploration, assumptions must be made about the sequence of data in the online interaction. For instance, Szepesvári [1998] assumes the data are IID (independent and identically distributed). A weaker kind of assumption is to assume that the MDP is fast mixing [Even-Dar and Mansour 2003b; Antos et al. 2008]. Under the mixing assumption, the exploration/exploitation dilemma is less challenging since the agent will have the chance to visit almost all states very quickly in a fast mixing MDP no matter what policy it uses.

Finally, Şimşek and Barto [2006] studied a different model for exploration, where the online interaction of the agent is separated into an exploration phase and an exploitation phase, similar to the training and testing separation in supervised learning. The goal of the agent is to collect as much information of the unknown MDP during exploration to compute a good policy for late exploitation. In contrast, the focus of the dissertation is more natural for a reinforcement-learning agent that has to balance exploration and exploitation *simultaneously*.

4.4.4 Sample Complexity of Exploration and PAC-MDP

This subsection defines the *sample complexity of exploration*, first proposed by Kakade [2003]. Such a notion is very general, without restrictive assumptions such as mixing that are needed in other notions of learning complexity. On the other hand, the criterion

of achieving small sample complexity of exploration is tractable in many challenging problems, and families of algorithms have been developed for various classes of MDPs. It is the main theme of the dissertation to create and analyze algorithms with polynomial sample complexity of exploration.

Definition

In measuring the sample complexity of exploration (or *sample complexity* for short) of a reinforcement-learning algorithm $\mathbf{A}$, we treat $\mathbf{A}$ as a non-stationary policy that maps histories to actions (*c.f.*, §2.2). Two additional inputs are needed: $\epsilon, \delta \in (0, 1)$. The *precision parameter*, ϵ, controls the quality of behavior we require of the algorithm (*i.e.*, how close to optimality do we desire to be). The *confidence parameter*, δ, measures how certain we want to be of the algorithm's performance. As both parameters decrease, greater exploration and learning is necessary as more is expected of the algorithms.

Similar to other learning algorithms for stochastic problems, we cannot expect an algorithm to identify the optimal policy with finite experience and with full confidence: we allow $\epsilon > 0$ because a finite sample does not reveal the whole dynamics of the MDP in general; we allow $\delta > 0$ because there is always some, albeit tiny, probability that the experience observed by the agent is not representative and thus misleads the agent to end up with a poor policy.

The following definition is proposed by Kakade [2003].

Definition 7 *Let $c_t = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_t)$ be a path generated by executing a reinforcement-learning algorithm $\mathbf{A}$ in an MDP M up to timestep t. The algorithm is viewed as a non-stationary policy, denoted $\mathbf{A}_t$ at timestep t. For any fixed $\epsilon > 0$, the* sample complexity *of $\mathbf{A}$ is the number of timesteps t such that the policy at time t, $\mathbf{A}_t$, is non-ϵ-optimal: $V^{\mathbf{A}_t}(s_t) \leq V^*(s_t) - \epsilon$.*

The sample complexity of exploration of an algorithm $\mathbf{A}$ is defined using the infinite-length online interaction between an agent running $\mathbf{A}$ and an environment. If we view it a "mistake" when $V^{\mathbf{A}_t}(s_t) \leq V^*(s_t) - \epsilon$ happens, then sample complexity of exploration measures the total number of mistakes of an RL algorithm during the whole run of it.

We place no limitations on when the algorithm makes a mistake during an infinite-length interaction with the MDP. This freedom is essential for RL as, in a stochastic MDP, the agent does not have absolute control of the sequence of states it will visit, and so a mistake may happen at any time during the interaction, depending on when the agent is stochastically transitioned to those "bad" states. Intuitively, we prefer algorithms with smaller sample complexity, which motivates the following notation of sample efficiency [Strehl et al. 2006a;c]:

Definition 8 *An algorithm* $\mathbf{A}$ *is said to be* PAC-MDP (Probably Approximately Correct in Markov Decision Processes) *in an MDP* M *if, for any* $\epsilon > 0$ *and* $0 < \delta < 1$*, the sample complexity of* $\mathbf{A}$ *is bounded by some function polynomial in the relevant quantities (*$1/\epsilon$, $1/\delta$, $1/(1-\gamma)$*, and* $|M|$*) with probability at least* $1-\delta$*, where* $|M|$ *measures the complexity of the MDP* M*. Usually,* $|M|$ *is polynomially related to the number of parameters describing* M*; for instance, it is the numbers of states and actions for finite MDPs, and it is the number of parameters if the MDP can be described in a parametric way. Furthermore, the algorithm is called* efficiently PAC-MDP *if its computational and sample complexities are polynomial.*

The terminology, PAC, in our definition is borrowed from Angluin [1988] for the distribution-free supervised-learning model proposed by Valiant [1984]. A discussion of the parameters involved in Definition 8 is in order:

- The precision parameter, ϵ, specifies the level of optimality we want the algorithm to achieve, and a mistake is incurred when the non-stationary policy, $\mathbf{A}_t$, is not ϵ-optimal. As in supervised learning, we cannot guarantee $\epsilon = 0$ because any finite set of samples does not reveal the complete dynamics of an MDP when it is stochastic.
- The confidence parameter, δ, measures how certain we want to be of the algorithm's sample complexity of exploration. As in supervised learning, we cannot guarantee $\delta = 0$ because there is always a non-zero, albeit tiny, probability that the observed interactions/samples are not representative of the true dynamics of the MDP and thus misleads the agent to act sub-optimally.

- The dependence on $1/(1-\gamma)$ is necessary and unique in reinforcement learning. In many RL analyses, $1/(1-\gamma)$ plays a role that is roughly the effective depth of decision horizon for value functions and policies (*c.f.*, Lemma 2). The larger this effective decision horizon is, the more we expect the sample complexity to be.
- The quantity $|M|$ of an MDP M is used to measure how large the MDP is. An analogue in PAC learning of concept classes is the number of variables in the Boolean formula [Valiant 1984]. The quantity $|M|$ is often defined in a natural way, and will be made clear when we analyze the sample complexity of exploration in later chapters.

Efficiency in the PAC-MDP framework implies efficiency in a more complicated framework called *average loss* which, as an analogue of per-timestep *regret* in sequential decision making problems, measures the actual per-timestep reward achieved by the agent against the expected per-timestep reward of the optimal policy in states the agent visits [Strehl and Littman 2008a]. While these two notions are closely related, we choose to focus on PAC-MDP learnability due to its cleanness.

The definition of sample complexity of exploration was first used in the analysis of Rmax by Kakade [2003], and will be the focus of the present dissertation. However, we note the analyses of E^3 by Kearns and Singh [2002], factored E^3 by Kearns and Koller [1999], and metric E^3 by Kakade et al. [2003] use slightly different definitions of efficient learning. In particular, these algorithms are required to halt after a polynomial amount of time and output a near-optimal policy for the last state, with high probability. Our analyses are essentially equivalent, but simpler in the sense that mixing-time arguments are avoided.

A General PAC-MDP Theorem

Most of the PAC-MDP results developed in this dissertation depend on the following general theorem for proving PAC-MDP-ness, first established by Strehl et al. [2006a] and then slightly generalized in later works [Strehl 2007b; Strehl et al. 2009]. The one presented here generalizes and improves previous versions and the modifications are important for our discussion in later chapters.

Definition 9 *During an online interaction as described in Definition 1, let K_t be a set of state–action pairs defined arbitrarily in a way that depends only on the history of the agent up to timestep t (before the t-th action is chosen). An* escape event*, denoted A_K, occurs at timestep t if $(s_t, a_t) \notin K_t$.*

Definition 10 *[Strehl et al. 2006a] Let $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ be an MDP, $Q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ a state–action value function, and $K \subseteq \mathcal{S} \times \mathcal{A}$ a subset of state–action pairs. We define the* known state–action MDP (with respect to K) *as $M_K = \langle \mathcal{S}, \mathcal{A}, T_K, R_K, \gamma \rangle$, where for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$,*

$$T_K(s' \mid s, a) \stackrel{\text{def}}{=} \begin{cases} T(s' \mid s, a), & \text{if } (s, a) \in K \\ \mathbb{I}(s = s'), & \text{otherwise,} \end{cases}$$

and

$$R_K(s, a) \stackrel{\text{def}}{=} \begin{cases} R(s, a), & \text{if } (s, a) \in K \\ (1 - \gamma) Q(s, a), & \text{otherwise.} \end{cases}$$

In words, the known state–action MDP is defined in a way so that the transition and reward functions coincide with the true MDP in state–action pairs in K, and the Q-functions are exact in state–action pairs outside K.

Theorem 4 *[Strehl et al. 2006a] Let $\mathbf{A}(\epsilon, \delta)$ be an algorithm that takes ϵ and δ as inputs (in addition to other algorithm-specific inputs), acts greedily according to its estimated state–action value function, denoted Q_t at timestep t. Define $V_t(s) \stackrel{\text{def}}{=} \text{argmax}_{a \in \mathcal{A}} Q_t(s, a)$. Suppose that on every timestep t, there exists a set K_t of state–action pairs that depends only on the agent's history up to timestep t. We assume that $K_t = K_{t+1}$ unless, during timestep t, an update to some state–action value occurs or the escape event A_K happens. Let M_{K_t} be the known state–action MDP and π_t be the greedy policy with respect to Q_t. Suppose that for any inputs ϵ and δ, with probability at least $1 - \delta/2$, the following conditions hold for all timesteps t:*

1. *(Optimism) $V_t(s_t) \geq V^*(s_t) - \epsilon/4$,*
2. *(Accuracy) $V_t(s_t) - V^{\pi_t}_{M_{K_t}}(s_t) \leq \epsilon/4$, and*

3. *(Bounded Surprises) The total number of updates of action–value estimates plus the number of times the escape event from K_t, A_K, can occur is bounded by a function $\zeta(\epsilon, \delta)$. The function ζ may depend on $|M|$.*

Then, with probability at least $1 - \delta$, the sample complexity of exploration of $\mathbf{A}(\epsilon, \delta)$ is

$$\mathcal{O}\left(\frac{V_{\max}}{\epsilon(1-\gamma)}\left(\zeta(\epsilon,\delta) + \ln\frac{1}{\delta}\right)\ln\frac{1}{\epsilon(1-\gamma)}\right).$$

Proof. Suppose that the learning algorithm $\mathbf{A}(\epsilon, \delta)$ is executed on an MDP M. Fix the history of the agent up to the t-th timestep. Define

$$H = \frac{1}{1-\gamma}\ln\frac{4}{\epsilon(1-\gamma)}.$$

Let W denote the event that, after following the policy $\mathbf{A}_t$ from state s_t in M for H timesteps, one of the two following events occur: (a) the algorithm performs a successful update (a change to its state–action value function) of some state–action pair (s, a), or (b) some state–action pair $(s, a) \notin K_t$ is experienced (escape event A_K). Then,

$$\begin{aligned}
V_M^{\mathbf{A}_t}(s_t, H) &\geq V_{M_{K_t}}^{\pi_t}(s_t, H) - \Pr(W)V_{\max} \\
&\geq V_{M_{K_t}}^{\pi_t}(s_t) - \frac{\epsilon}{4} - \Pr(W)V_{\max} \\
&\geq V(s_t) - \frac{\epsilon}{2} - \Pr(W)V_{\max} \\
&\geq V^*(s_t) - \frac{3\epsilon}{4} - \Pr(W)V_{\max}.
\end{aligned}$$

where: the first step follows from the fact that following $\mathbf{A}_t$ in MDP M is identical to following π_t in M_{K_t} unless event W occurs, in which case a difference of at most $V_{\max}$ can occur; the second step is due to Lemma 2; and the last two steps make use of the accuracy and optimism conditions.

Now, suppose $\Pr(W) < \frac{\epsilon}{4V_{\max}}$ and the inequality derived above implies ϵ-optimality of the non-stationary policy at timestep t:

$$V_M^{\mathbf{A}_t}(s_t) \geq V_M^{\mathbf{A}_t}(s_t, H) \geq V^*(s_t) - \epsilon.$$

Otherwise, $\Pr(W) \geq \frac{\epsilon}{4V_{\max}}$; that is, an agent following $\mathbf{A}_t$ will either update its state–action value function in H timesteps, or encounter some $(s, a) \notin K_t$ in H timesteps,

with probability at least $\frac{\epsilon}{4V_{\max}}$. We view this H-step trajectory as a Bernoulli trial with success probability at least $\frac{\epsilon}{4V_{\max}}$. Then, Lemma 56 implies that after

$$\frac{8HV_{\max}}{\epsilon}\left(\zeta(\epsilon,\delta)+\ln\frac{2}{\delta}\right)=\mathcal{O}\left(\frac{V_{\max}}{\epsilon(1-\gamma)}\left(\zeta(\epsilon,\delta)+\ln\frac{1}{\delta}\right)\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

timesteps t where $\Pr(W)\geq\frac{\epsilon}{4V_{\max}}$, $\zeta(\epsilon,\delta)$ successes will occur, with probability at least $1-\delta/2$. However, by Condition 3, with high probability, $\zeta(\epsilon,\delta)$ is the maximum number of successes in these Bernoulli trials that can occur during an entire execution of $\mathbf{A}$.

Finally, we may complete the proof using a union bound to bound the total failure probability by $\delta/2+\delta/2=\delta$. □

Bayesian Reinforcement Learning Is Not PAC-MDP

Given the fact that Bayesian exploration maximizes the expected utility in every action, it is natural to ask how it is related to the PAC-MDP framework. The following example shows a Bayesian RL algorithm is not PAC-MDP in general. In fact, the example shows a stronger assertion that Bayesian RL in not consistent: the learned policy may not converge to the optimal policy even in the limit. A more general, yet more complicated, result is given by Kolter and Ng [2009a] that applies to any greedy algorithm with a fast decaying exploration bonus:

Example 9 *The MDP is a two-armed bandit; namely, it has a single state and two actions with self-looping transitions (Figure 4.2). The reward of action a_1 is known to be $1/2$; equivalently, the prior probability of $R(a_1)$ being $1/2$ is 1. The reward of action a_2 may have two values, whose respective prior probabilities are:*

$$\begin{aligned}\Pr\left(R(a_2)=0\right) &= p\\ \Pr\left(R(a_2)=1/2+\epsilon(1-\gamma)\right) &= 1-p,\end{aligned}$$

for some $p\in\mathbb{R}_+$ that we will specify. If $R(a_2)=0$, the optimal policy is to choose a_1; otherwise, choosing a_1 is not an ϵ-optimal policy as the difference between this policy and the optimal policy that chooses a_2 is $\epsilon(1-\gamma)(1+\gamma+\gamma^2+\cdots)=\epsilon$. Since the MDP has only one state, we suppress dependence on states, and thus the prior and posterior distributions of reward functions are represented as a binomial distribution:

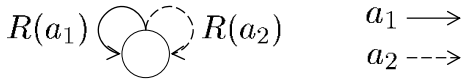


Figure 4.2: An example showing Bayesian exploration may not be PAC-MDP or consistent.

$b = (\alpha, 1-\alpha)$*, where* $\alpha \in [0,1]$ *is the probability that* $R(a_2) = 0$*. Initially,* $b_0 = (p, 1-p)$*. The Bellman equation of this MDP in "belief state"* b_0 *is*

$$\begin{aligned} Q^*(b_0, a_1) &= R(a_1) + \gamma V^*(b_0) \\ Q^*(b_0, a_2) &= p\left(\gamma V^*(b_1)\right) + (1-p)\left(\frac{1}{2} + \epsilon(1-\gamma) + \gamma V^*(b_2)\right), \end{aligned}$$

where $b_1 = (1,0)$ *and* $b_2 = (0,1)$ *correspond to the next belief states after action* a_2 *is chosen. Since complete knowledge of the MDP is available with belief state* b_1 *or* b_2*, we can compute their value functions easily:*

$$\begin{aligned} V^*(b_1) &= \frac{1}{2(1-\gamma)} \\ V^*(b_2) &= \frac{1}{2(1-\gamma)} + \epsilon. \end{aligned}$$

Our goal is to decide the free parameter p *such that* $Q^*(b_0, a_1) > Q^*(b_0, a_2)$*, and so a Bayesian algorithm will* never *choose action* a_2*, and thus will fail to find the optimal policy when* $R(a_2) = 1/2 + \epsilon(1-\gamma)$*.*

Instead of computing $V^*(b_0)$ *exactly using the Bellman equation above, it suffices for our purpose to show that* $Q^*(b_0, a_2)$ *is smaller than a lower bound of* $Q^*(b_0, a_1)$*. Specifically, we have*

$$Q^*(b_0, a_1) = \frac{1}{2} + \gamma V^*(b_0) \geq \frac{1}{2} + \gamma Q^*(b_0, a_1),$$

and thus

$$Q^*(b_0, a_1) \geq \frac{1}{2(1-\gamma)}.$$

On the other hand, we may compute $Q^*(b_0, a_2)$ *easily by the Bellman equation:*

$$\begin{aligned} Q^*(b_0, a_2) &= p\left(0 + \gamma V^*(b_1)\right) + (1-p)\left(\frac{1}{2} + \epsilon(1-\gamma) + \gamma V^*(b_2)\right) \\ &= -\frac{p}{2} + (1-p)\epsilon + \frac{1}{2(1-\gamma)}. \end{aligned}$$

Therefore,

$$Q^*(b_0, a_1) - Q^*(b_0, a_2) \quad \geq \quad p(\epsilon + 1/2) - \epsilon.$$

and it suffices to set

$$p > \frac{\epsilon}{\epsilon + 1/2} \approx 2\epsilon$$

to force a Bayesian algorithm to always select a_1*.*

In this example above, it is interesting to note that the undesirable inconsistency phenomenon happens for essentially *all* values of p, not just for extreme values. However, it is possible to introduce PAC-MDP-like result in Bayesian RL with techniques we study in this dissertation [Asmuth et al. 2009].

4.5 Use of Models in Reinforcement Learning

Most reinforcement-learning algorithms can roughly be categorized into two families: *model-based* and *model-free*. As mentioned in §4.1, model-based algorithms work by estimating parameters of the unknown MDP and then computing the optimal policy of the estimated MDP model. The simulation lemma insures that the policy will be near-optimal as long as the MDP is learned to sufficient accuracy. Model-free algorithms, on the other hand, often learn the optimal value function directly without trying to learn the MDP model, either explicitly or implicitly.

The use of model estimation in reinforcement learning has been a topic of controversy (see, *e.g.*, Atkeson and Gordon [1997]). While there are good reasons to prefer model-based approaches, other concerns make model-free approaches a better choice. In our following discussions, the term "model-based" is used in a broad sense which either means (i) the algorithm builds an explicit model such as in certainty equivalence, or (ii) the algorithm uses an *implicit* representation of a model such as experience replay.

The major reason to use model-based approaches is probably that they are often more sample-efficient—a critical advantage in situations where data are expensive, such as in robotics [Sutton 1990; Lin 1992; Moore and Atkeson 1993; Peng and Williams 1993; Atkeson and Santamaria 1997]. For instance, Sutton [1990] proposes the Dyna architecture with two instantiations: Dyna-PI and Dyna-Q. In every timestep, the learned MDP

model is used to generate hypothetical state transitions, with which Bellman backups are performed (called *relaxation planning*). In contrast, model-free algorithms such as Q-learning often discard a sample transition after performing a stochastic-approximation step with it, and so are generally considered less sample-efficient.

A related advantage of model-based approaches is that they can facilitate efficient exploration [Moore and Atkeson 1993; Schneider 1997; Jong and Stone 2007]. By explicitly estimating a model and measuring uncertainty in the model parameters, an agent is able to tell which part of the state space needs more exploration. In fact, the first algorithm with provably efficient exploration in general, finite MDPs is model-based [Kearns and Singh 2002]. In contrast, efficient exploration is harder to achieve in model-free approaches. Simple heuristics like ϵ-greedy can be inefficient (Example 8), and rather complicated exploration strategies are needed [Strehl et al. 2006c].

But there are limitations of model-based approaches, which make model-free approaches a better choice in some cases. A major problem with model-based methods is that they are often computational expensive. Solving an MDP often suffers the *curse of dimensionality* [Bellman 1957]: the computational complexity of finding a (near) optimal policy in an MDP often grows exponentially with the number of variables used to describe a state [Chow and Tsitsiklis 1989]. Approximate planning algorithms with policy-quality guarantees may suffer an exponential dependence on other quantities (*c.f.*, §3.3.4).

A more subtle risk with model-based approaches is that an agent can end up with suboptimal policies when the model is misleadingly imprecise. For example, Lin [1992] found empirically that model building may actually slow down learning. Kuvayev and Sutton [1997] also report empirical evidence that using a learned but imprecise model in fact impedes learning in a Dyna-style algorithm. Recently, a hybrid approach is proposed so that the learned, but possibly imprecise, model is used to make local improvement to the policy [Abbeel et al. 2006b].

The results in the dissertation provide novel insights regarding comparisons between model-based and model-free approaches.

Part II

KWIK Learning

Chapter 5

The KWIK Learning Framework

This and the next chapters deviate from reinforcement learning and focus on a new *supervised*-learning framework called *Knows What It Knows*, or *KWIK* in short [Li et al. 2008]. This framework provides a unifying mathematical background for the formal study of efficient RL algorithms in Part III.

5.1 Definition

As shown by Example 7, the key property of a KWIK learner provides a useful mechanism for efficiently exploring an unknown environment. We formalize the learning model here. A KWIK problem is specified by a five-tuple: $P = \langle \mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, h^* \rangle$, where: $\mathcal{X}$ is an *input set*; $\mathcal{Y}$ is an *output set*; $\mathcal{Z}$ is a *observation set*; $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ is a *hypothesis set* consisting of hypotheses mapping $\mathcal{X}$ to $\mathcal{Y}$; and $h^* \in \mathcal{Y}^{\mathcal{X}}$ is an unknown *target function*.

In this dissertation, we make the following "realization assumption" for KWIK problems:

Assumption 1 *In a KWIK problem $P = \langle \mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, h^* \rangle$, we have $h^* \in \mathcal{H}$.*

The study of "unrealizable" KWIK models are left for future work; see §5.5 for a preliminary discussion.

Two parties are involved in a KWIK learning process. The *learner* runs a learning algorithm and makes predictions; the *environment*, which represents an instance of a KWIK learning problem, provides the learner with inputs and observations. The protocol for a KWIK "run" is as follows (Figure 5.1):

1. The input set $\mathcal{X}$, output set $\mathcal{Y}$, observation set $\mathcal{Z}$, hypothesis class $\mathcal{H}$, accuracy parameter $\epsilon \in (0, 1)$, and confidence parameter $\delta \in (0, 1)$ are known to both the

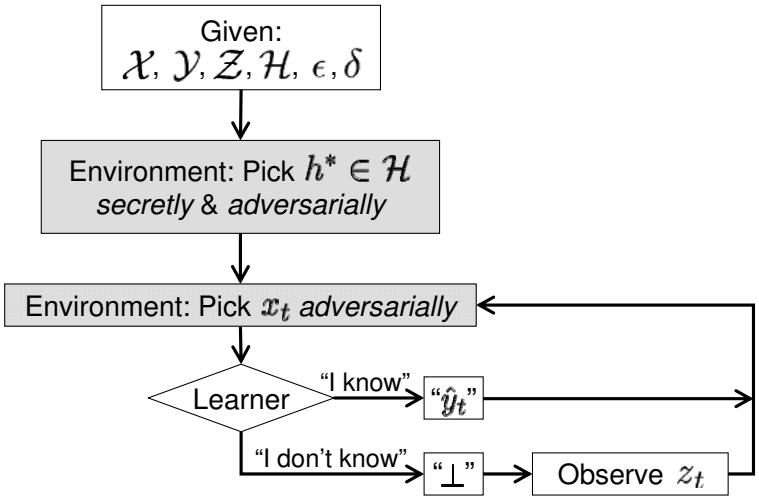


Figure 5.1: KWIK protocol.

learner and the environment.

2. The environment selects a target function $h^* \in \mathcal{H}$ *secretly* and *adversarially*.
3. At timestep $t = 1, 2, 3, \ldots$

   - The environment selects an input $x_t \in \mathcal{X}$ in an *arbitrary* way and informs the learner. The target value $y_t = h^*(x_t)$ is unknown to the learner.
   - The learner predicts an output $\hat{y}_t \in \mathcal{Y} \cup \{\perp\}$, where $\perp$ is a special symbol that is not in $\mathcal{Y}$. We call $\hat{y}_t$ *valid* if $\hat{y} \neq \perp$.
   - If $\hat{y}_t = \perp$, the learner makes an observation $z_t \in \mathcal{Z}$ of the output. In the deterministic case, $z_t = y_t$. In the stochastic case, the relation between z_t and y_t is problem specific, and may or may not be known by the learner. For example, §5.3.1 considers a problem where $z_t = 1$ with probability y_t and 0 with probability $1 - y_t$ when $\mathcal{Z} = \{0, 1\}$, and §5.3.3 assumes $z_t = y_t + \eta_t$ where η_t is a normally distributed random variable with zero mean and known variance, while more complicated examples arise in §6.

To simplify notation, we always assume that $\mathcal{Y}$ is a subset of $\mathbb{R}^n$ for an appropriate value of n, so that the subtraction, $y_1 - y_2$, is well-defined for any elements $y_1, y_2 \in \mathcal{Y}$. With this numeric representation of $\mathcal{Y}$, we further require that $\mathcal{Y}$ be equipped with a real-valued function, $|\cdot|$, so that $|y_1 - y_2|$ measures the *discrepancy* or *distance* between

y_1 and y_2. For example, $|\cdot|$ may be understood to be a vector ℓ_p-norm. Conceptually, any real-valued function can be used in the KWIK protocol above. However, certain conditions are necessary for a formal analysis of algorithms in this and the next chapters. Specifically, we make the following mild assumptions, which, for instance, are satisfied when $|\cdot|$ is a semi-norm:

Assumption 2 *The discrepancy function, $|\cdot| \in \mathbb{R}^{\mathcal{Y}}$, associated with $\mathcal{Y}$ satisfies the following conditions: for all $y, y_1, y_2 \in \mathcal{Y}$,*

- $|y - y| = 0$,
- *Non-negativity:* $|y_1 - y_2| \geq 0$,
- *Symmetry:* $|y_1 - y_2| = |y_2 - y_1|$, *and*
- *Triangle inequality:* $|y_1 - y_2| \leq |y_1 - y| + |y - y_2|$.

Definition 11 *Let $P = \langle \mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, h^* \rangle$ be a KWIK problem. We say that $\mathcal{H}$ is* KWIK-learnable *if there exists an algorithm* **A** *with the following property: for any $0 < \epsilon, \delta < 1$, the following two requirements are satisfied with probability at least $1 - \delta$ in a whole run of* **A** *according to the KWIK protocol above:*

1. *(*Accuracy*) If $\hat{y}_t \neq \perp$, it must be ϵ-accurate: $|\hat{y}_t - y_t| < \epsilon$;*
2. *(*Sample complexity*) The total number of $\perp$s predicted during the whole run is bounded by a function, denoted $B(\epsilon, \delta, \dim(\mathcal{H}))$, which is polynomial in $1/\epsilon$, $1/\delta$, and $\dim(\mathcal{H})$, where $\dim(\mathcal{H})$ is a pre-defined nonnegative-valued function measuring the* dimension *or* complexity *of $\mathcal{H}$.*

We call **A** *a* KWIK algorithm *and $B(\epsilon, \delta, \dim(\mathcal{H}))$ its* KWIK bound. *We will also use $B(\epsilon, \delta)$ for simplicity, if there is no ambiguity. Furthermore, $\mathcal{H}$ is* efficiently KWIK-learnable *if, in addition to the accuracy and sample-complexity requirements, the per-timestep computational complexity of* **A** *is polynomial in $1/\epsilon$, $1/\delta$, and $\dim(\mathcal{H})$.*

All KWIK algorithms studied in this dissertation enjoy polynomial computational complexity and thus are efficient. We note that a KWIK algorithm can be randomized, and so the confidence parameter, δ, may be necessary even in deterministic problems.

Finally, we point out a minor difference between the above definition and the original one. Li et al. [2008] requires a KWIK algorithm must produce finitely many $\perp$s with

probability 1, while Definition 11 allows infinitely many $\perp$s with at most δ probability. To show these two definitions are equivalent, it suffices to show that any KWIK algorithm $\mathbf{A}$ satisfying Definition 11 with a KWIK bound $B(\epsilon, \delta, \dim(\mathcal{H}))$ can be converted into another algorithm $\mathbf{A}'$ satisfying the definition in Li et al. [2008]. The conversion is straightforward: $\mathbf{A}'$ simply simulates $\mathbf{A}$ using parameter ϵ and δ, and then makes arbitrary predictions if $\mathbf{A}$ returns more than $B(\epsilon, \delta, \dim(\mathcal{H}))$ many $\perp$s. By Definition 11, the probability that $\mathbf{A}$ outputs more than $B(\epsilon, \delta, \dim(\mathcal{H}))$ many $\perp$s or makes a non-ϵ-accurate prediction during a run is at most δ. It follows that $\mathbf{A}'$ dissatisfies the accuracy requirement with probability at most δ. Furthermore, the number of $\perp$s returned by $\mathbf{A}'$ is at most $B(\epsilon, \delta, \dim(\mathcal{H}))$, by construction. Therefore, $\mathbf{A}'$ satisfies the KWIK definition of Li et al. [2008].

5.2 Example KWIK Learners in Deterministic Problems

This and the next sections describe some KWIK-learnable hypothesis classes, some of which will be useful in later chapters. We begin by describing the simplest and most general KWIK algorithms, and then transition to more complicated, stochastic problems in the next section.

5.2.1 Memorization

The memorization algorithm (Algorithm 11) can be used when the input set is finite and observations are deterministic.

The algorithm simply memorizes the corresponding output for each possible $x \in \mathcal{X}$. It initializes a hypothesis, $\hat{h}$, that is represented by a lookup table and predicts $\perp$ for all inputs. When the environment chooses an input x for the first time, the algorithm reports $\perp$ to observe the corresponding y and remembers it by storing y in entry $\hat{h}(x)$. It only reports $\perp$ once for each input. We have thus proved the following theorem:

Theorem 5 *Memorization is a KWIK algorithm for deterministic KWIK problems, $P = \langle \mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, h^* \rangle$, where $\mathcal{X}$ is finite, with the following KWIK bound:*

$$B_{\text{memorization}}(\epsilon, \delta) = |\mathcal{X}| .$$

While memorization ignores ϵ and δ, the KWIK bound in Theorem 5 is clearly tight. The computational complexity of the algorithm is linear in the time required for retrieving entries in the table $\hat{h}$, which is constant if it is a hash table and is at most $\mathcal{O}(|\mathcal{X}|)$ with a reasonable representation of $\hat{h}$.

Algorithm 11 Memorization.

0: **Inputs:** $\mathcal{X}, \mathcal{Y}, \mathcal{H}, \epsilon, \delta$.
1: Initialize $\hat{h}(x) \leftarrow \perp$ for all $x \in \mathcal{X}$.
2: **for** $t = 1, 2, \ldots$ **do**
3: Observe $x_t \in \mathcal{X}$ and predict $\hat{y}_t = \hat{h}(x)$.
4: **if** $\hat{h}(x) = \perp$ **then**
5: Observe y_t.
6: Update hypothesis: $\hat{h}(x) \leftarrow y_t$.
7: **end if**
8: **end for**

5.2.2 Enumeration

When the hypothesis set is finite and observations are deterministic, the enumeration algorithm (Algorithm 12) can be used.

The algorithm keeps track of $\hat{\mathcal{H}} \subseteq \mathcal{H}$, the *version space* containing hypotheses that are consistent with observed data. Initially, $\hat{\mathcal{H}}$ is set to $\mathcal{H}$. In every timestep t, the algorithm computes $\hat{L} = \{h(x_t) \mid h \in \hat{\mathcal{H}}\}$, the set of outputs for x_t computed by all hypotheses in the version space. If $\left|\hat{L}\right| = 0$, then the version space is empty, implying that the target hypothesis is not in the hypothesis class $\mathcal{H}$, which violates Assumption 1. If $\left|\hat{L}\right| = 1$, it means that all remaining hypotheses in $\hat{\mathcal{H}}$ agree on the output for this input, and therefore the algorithm knows what the correct output must be, thanks to the realization assumption. In this case, the only element in $\hat{L}$ is predicted as the output. On the other hand, if $\left|\hat{L}\right| > 1$, two hypotheses in the version space disagree. Hence, the algorithm is uncertain about the true output and has to return $\perp$ to receive the true label y_t. It then updates the version space by eliminating all mistaken hypotheses (Line 9 of Algorithm 12). Therefore, the new version space must be strictly smaller. If $\left|\hat{\mathcal{H}}\right| = 1$ at any point, $\left|\hat{L}\right| = 1$ for all future inputs, and the algorithm will no longer return $\perp$. We have thus proved the main theorem in this section:

Theorem 6 *Enumeration is a KWIK algorithm for deterministic KWIK problems, $P = \langle \mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, h^* \rangle$, where $|\mathcal{H}|$ is finite, with the following KWIK bound:*

$$B_{\textsf{enumeration}}(\epsilon, \delta) = |\mathcal{H}| - 1.$$

The computational complexity of enumeration is $\mathcal{O}(|\mathcal{H}|)$, assuming it takes $\mathcal{O}(1)$ time to compute $h(x)$ for all $h \in \mathcal{H}$ and $x \in \mathcal{X}$.

Algorithm 12 Enumeration.

Inputs: $\mathcal{X}, \mathcal{Y}, \mathcal{H}, \epsilon, \delta$.
$\hat{\mathcal{H}} \leftarrow \mathcal{H}$.
for $t = 1, 2, \ldots$ **do**
  Observe $x_t \in \mathcal{X}$.
  Compute $\hat{L} = \{h(x_t) \mid h \in \hat{\mathcal{H}}\}$.
  if $\left|\hat{L}\right| = 1$ **then**
    Predict $\hat{y}_t = y$ where y is the only element in $\hat{L}$.
  else
    Predict $\hat{y}_t = \perp$ and observe y_t.
    $\hat{\mathcal{H}} \leftarrow \hat{\mathcal{H}} \setminus \{h \mid h \in \hat{\mathcal{H}} \wedge h(x_t) \neq y_t\}$.
  end if
end for

An example given by Li et al. [2008] shows enumeration may enjoy an exponentially smaller KWIK bound than memorization. However, memorization becomes better when $\mathcal{H}$ is large or infinite.

5.2.3 Deterministic Linear Regression

The previous two algorithms exploited the finiteness of the hypothesis class and input set. Finite KWIK bounds can also be achieved when these sets are all infinite, as shown by the deterministic linear regression problem. Specifically, we define $\mathcal{X} = \mathbb{R}^n$, $\mathcal{Y} = \mathcal{Z} = \mathbb{R}$, and

$$\mathcal{H} = \{f_{\mathbf{w}} \mid f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}, \mathbf{w} \in \mathbb{R}^n\}.$$

That is, $\mathcal{H}$ is the set of linear functions on n variables for some unknown weight vector $\mathbf{w}$. In the deterministic case where $z_t = y_t$, $\mathcal{H}$ can be KWIK-learned by the deterministic linear-regression algorithm (Algorithm 13) with the following guarantee:

Algorithm 13 Deterministic linear-regression.

Inputs: $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, \epsilon, \delta$.
$\mathcal{D} \leftarrow \emptyset$.
for $t = 1, 2, \ldots$ **do**
  Let $\mathcal{D}$ be $\{(\mathbf{v}_1, f(\mathbf{v}_1)), (\mathbf{v}_2, f(\mathbf{v}_2)), \ldots, (\mathbf{v}_k, f(\mathbf{v}_k))\}$.
  Observe $\mathbf{x}_t \in \mathcal{X}$.
  if $\mathbf{x}$ is linearly independent of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ **then**
    Predict $\hat{y}_t = \perp$ and observe y_t.
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_t, y_t)\}$.
  else
    Let $\mathbf{x}_t = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k$ for some $a_1, \ldots, a_k \in \mathbb{R}$.
    Predict $\hat{y}_t$ using Equation 5.1.
  end if
end for

Theorem 7 *Deterministic linear-regression solves the deterministic linear-regression problem with a KWIK bound of:*

$$B_{\text{deterministic linear-regression}}(\epsilon, \delta) = n.$$

PROOF. The algorithm maintains a training set $\mathcal{D}$ of training examples, which is initialized to the empty set $\emptyset$ prior to learning. On the t-th input $\mathbf{x}_t$, let

$$\mathcal{D} = \{(\mathbf{v}_1, f(\mathbf{v}_1)), (\mathbf{v}_2, f(\mathbf{v}_2)), \cdots, (\mathbf{v}_k, f(\mathbf{v}_k))\}$$

be the current set of training examples. The algorithm first detects if x_t is linearly dependent of the previous inputs stored in $\mathcal{D}$. This test can be decided efficiently by, say, Gaussian elimination [Golub and Van Loan 1996]. If $\mathbf{x}_t$ is linearly independent, then the algorithm predicts $\perp$, observes the output $y_t = f(\mathbf{x}_t)$, and then expands the training set: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_t, y_t)\}$. Otherwise, there exist k real numbers, $a_1, a_2, \cdots, a_k$, such that $\mathbf{x}_t = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_k\mathbf{v}_k$. In this case, we can accurately predict the value of $f(\mathbf{x}_t)$ using information in $\mathcal{D}$, without necessarily knowing the weight vector $\mathbf{w}$:

$$\begin{aligned} f(\mathbf{x}_t) &= \mathbf{w}^\top \mathbf{x}_t \\ &= \mathbf{w}^\top (a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_k\mathbf{v}_k) \\ &= a_1\mathbf{w}^\top\mathbf{v}_1 + a_2\mathbf{w}^\top\mathbf{v}_2 + \ldots + a_k\mathbf{w}^\top\mathbf{v}_k \\ &= a_1 f(\mathbf{v}_1) + a_2 f(\mathbf{v}_2) + \ldots + a_k f(\mathbf{v}_k). \end{aligned} \tag{5.1}$$

Thus, the accuracy requirement is satisfied.

By operation of the algorithm, the inputs in $\mathcal{D}$ (*i.e.*, $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$) must be linearly independent at all times. Hence, $\mathcal{D}$ contains at most n training pairs. Since every $\perp$ will increase the size of $\mathcal{D}$ by 1, the algorithm can return $\perp$ at most n times. $\square$

Any case where the sequence of inputs, $[\mathbf{x}_t]_{t \in \mathbb{N}}$, contain n linearly independent vectors may be used to show this KWIK bound is tight. Furthermore, we may allow a small prediction error if $\mathbf{x}_t$ is "almost" linearly dependent on the inputs in $\mathcal{D}$, which may result in practically more useful algorithms. However, we shall confine ourselves to a slightly more constrained hypothesis class where the weight vector has a bounded ℓ_2-norm:

$$\mathcal{H} = \{f_{\mathbf{w}} \mid f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}, \mathbf{w} \in \mathbb{R}^n, \|\mathbf{w}\|_2 \leq 1\}.$$

We decompose $\mathbf{x}_t$ into a linear combination of $\mathbf{v}_1, \ldots, \mathbf{v}_k$ plus a residual term $\Delta\mathbf{x}_t$ that is orthogonal to all $\mathbf{v}_i$:

$$\mathbf{x}_t = a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k + \Delta\mathbf{x}_t.$$

If $\|\Delta\mathbf{x}_t\|_2 \leq \epsilon$, we may pretend the input is

$$\tilde{\mathbf{x}}_t = \mathbf{x}_t - \Delta\mathbf{x}_t = a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k$$

and apply Equation 5.1 to predict $f(\tilde{\mathbf{x}}_t)$ precisely. The prediction, $\hat{y}_t \leftarrow f(\tilde{\mathbf{x}}_t)$, must be ϵ-close to $f(\mathbf{x}_t)$:

$$\begin{aligned}
|\hat{y}_t - f(\mathbf{x}_t)| &= \left|\hat{y}_t - \mathbf{w}^\top\tilde{\mathbf{x}}_t + \mathbf{w}^\top\tilde{\mathbf{x}}_t - \mathbf{w}^\top\mathbf{x}_t\right| \\
&\leq \left|\hat{y}_t - \mathbf{w}^\top\tilde{\mathbf{x}}_t\right| + \left|\mathbf{w}^\top\tilde{\mathbf{x}}_t - \mathbf{w}^\top\mathbf{x}_t\right| \\
&= 0 + \left|\mathbf{w}^\top\Delta\mathbf{x}_t\right| \\
&\leq \|\mathbf{w}\|_2 \|\Delta\mathbf{x}_t\|_2 \\
&\leq \epsilon,
\end{aligned}$$

where we have applied the triangle inequality in the first inequality and the Cauchy-Schwarz inequality in the second.

5.2.4 Distance Learning

This section considers another KWIK problem with infinite input/output sets and hypothesis class. In distance learning, we are to KWIK-learn the distance of an input point to an unknown point in a Euclidean space.

Define $\mathcal{X} = \mathbb{R}^n$, $\mathcal{Y} = \mathcal{Z} = \mathbb{R}_+ \cup \{0\}$, and

$$\mathcal{H} = \{f_{\mathbf{c}} \mid f_{\mathbf{c}}(\mathbf{x}) = \|\mathbf{x} - \mathbf{c}\|_2, \mathbf{c} \in \mathbb{R}^n\},$$

where $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^\top \mathbf{x}}$ denotes the ℓ_2-norm of vector $\mathbf{x} \in \mathbb{R}^n$. That is, there is an unknown point and the target function maps input points to the ℓ_2-distance from the unknown point. Since translations do not change the ℓ_2-distance metric in $\mathbb{R}^n$, we may assume without loss of generality that the first input is the origin: $\mathbf{x}_1 = \mathbf{0}$. This assumption will simplify our exposition; otherwise, we simply subtract $\mathbf{x}_1$ from all inputs $\mathbf{x}_t$. Although this problem can be solved using a geometric argument [Li et al. 2008], we give a simpler solution by reducing it to the deterministic linear regression problem that we studied in the previous subsection.[1]

First, consider the squared ℓ_2-distance of an input $\mathbf{x}$ to the unknown point $\mathbf{c}$:

$$\|\mathbf{x} - \mathbf{c}\|_2^2 = \|\mathbf{c}\|_2^2 - 2\mathbf{c}^\top \mathbf{x} + \|\mathbf{x}\|_2^2, \tag{5.2}$$

If we augment the input $\mathbf{x}$ by appending a unity component to it to yield a $(n+1)$-dimensional column vector, denoted $\mathbf{x}'$:

$$\mathbf{x}' \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{x}^\top, & 1 \end{bmatrix}^\top \in \mathbb{R}^{n+1}$$

then Equation 5.2 can be rewritten as

$$\|\mathbf{x} - \mathbf{c}\|_2^2 - \|\mathbf{x}\|_2^2 = \mathbf{w}^\top \mathbf{x}',$$

where

$$\mathbf{w} \stackrel{\text{def}}{=} \begin{bmatrix} -2\mathbf{c}^\top, & \|\mathbf{c}\|_2^2 \end{bmatrix}^\top \in \mathbb{R}^{n+1}$$

is an unknown weight vector defining a linear function in the augmented input $\mathbf{x}'$. Therefore, we can learn the following hypothesis class with a KWIK bound of $n+1$,

[1] This reduction is due to Robert Schapire.

based on Theorem 7:

$$\mathcal{H}' = \{g_{\mathbf{c}} \mid g_{\mathbf{c}}(\mathbf{x}) = \|\mathbf{x} - \mathbf{c}\|_2^2 - \|\mathbf{x}\|_2^2, \mathbf{c} \in \mathbb{R}^n\}.$$

Finally, if we can predict $g_{\mathbf{c}}(\mathbf{x})$ for an input $\mathbf{x}$ accurately, we can use the following formula to compute $f_{\mathbf{c}}(\mathbf{x})$ exactly:

$$f_{\mathbf{c}}(\mathbf{x}) = \sqrt{g_{\mathbf{c}}(\mathbf{x}) + \|\mathbf{x}\|_2^2}.$$

Thus, we have proved the main result of this subsection:

Theorem 8 *Distance-learning is a KWIK algorithm with the following KWIK bound:*

$$B_{\textit{distance-learning}}(\epsilon, \delta) = n + 1.$$

5.3 Example KWIK Learners in Stochastic Problems

In problems in the previous section, observations are noise free. In this section, we consider a few fundamental, noisy KWIK learning problems.

5.3.1 Coin Learning

We start with the simplest Bernoulli case and note that the same algorithm can actually be applied to KWIK-learn the expectation of a bounded, real-valued random variable.

Algorithm 14 Coin-learning.

Inputs: $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, \epsilon, \delta, m$.
$c \leftarrow 0$. {Number of observed data.}
$\hat{p} \leftarrow 0$. {Maximum-likelihood of p.}
for $t = 1, 2, \ldots$ **do**
  Observe $\mathbf{x}_t \in \mathcal{X}$.
  if $c < m$ **then**
    Predict $\hat{y}_t = \perp$ and observe z_t.
    $c \leftarrow c + 1$.
    $\hat{p} \leftarrow \hat{p} + (z_t - \hat{p})/c$.
  else
    Predict $\hat{y}_t = \hat{p}$.
  end if
end for

We have a biased coin whose unknown probability of heads is p. In the notation of this paper, $\mathcal{X}$ is a singleton containing an arbitrarily chosen element (meaning that we have a single coin with a fixed but unknown head probability), $\mathcal{Y} = \mathcal{H} = [0, 1]$, and $\mathcal{Z} = \{0, 1\}$ with 0 for tail and 1 for head. With probability at least $1 - \delta$, we want to learn an estimate $\hat{p}$ that is ϵ-accurate: $|\hat{p} - p| \leq \epsilon$. Note that $|\hat{p} - p|$ coincides with the total variation between two binomial distributions, $\mathrm{Bin}(\hat{p}, 1 - \hat{p})$ and $\mathrm{Bin}(p, 1 - p)$.

If we could observe p, then this problem would be trivial with a KWIK bound of 1. However, observations are noisy: instead of observing p, we see either 1 (with probability p) or 0 (with probability $1 - p$). A natural idea to make use of such discrete outcomes would be to compute the maximum-likelihood estimate given enough data. Each time the algorithm says $\perp$, it gets an independent trial that it can use to compute the empirical probability:

$$\hat{p} = \frac{1}{m} \sum_{t=1}^{m} z_t,$$

where $z_t \in \mathcal{Z}$ is the t-th observation in the m trials. The number of trials needed before we are $1 - \delta$ certain our estimate is within ϵ-accuracy follows directly from Hoeffding's inequality (Lemma 52):

$$m = \frac{1}{2\epsilon^2} \ln \frac{2}{\delta}.$$

We have thus proved the main theorem in this section:

Theorem 9 *With an appropriate parameter m, the* coin-learning *algorithm (Algorithm 14) can accurately learn a binomial distribution with a KWIK bound of*

$$B_{\textsf{coin-learning}}(\epsilon, \delta) = m = \frac{1}{2\epsilon^2} \ln \frac{2}{\delta} = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right).$$

Note that a trick is used in Line 8 of Algorithm 14 to maintain a running average of all observations seen so far, resulting in $\mathcal{O}(1)$ space complexity. The computational complexity of the algorithm is $\mathcal{O}(1)$, assuming all arithmetic operations can be done in constant time.

5.3.2 Dice Learning

Dice-learning generalizes coin-learning and can KWIK-learn a multinomial distribution over n elements, where each distribution is specified by n non-negative numbers that

sum up to unity. An example is dice rolling (hence the name of the algorithm) where one tries to predict the probabilities of getting $1, 2, \ldots, 6$; in this case, $n = 6$.

Formally, the input set $\mathcal{X}$ still contains a single element, meaning there is one specific multinomial distribution to learn; the output set and hypothesis class coincide and both contain all possible multinomial distributions over n elements:

$$\mathcal{Y} = \mathcal{H} = \{\mathbf{p} \mid \mathbf{p} \in \mathbb{R}_+^n, \|\mathbf{p}\|_1 = 1\};$$

and the observation set, for convenience, is the set of elementary basis vectors in the n-dimensional space: $\mathcal{Z} = \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n\}$, where $\mathbf{e}_i = [0, \ldots, 0, 1, 0, \ldots, 0]^\top$ with the only nonzero element 1 in the i-th component. Naturally, $\mathbf{e}_i$ corresponds to the discrete observation that the i-th outcome is observed. Given two discrete distributions in $\mathcal{H}$, $\mathbf{p}$ and $\hat{\mathbf{p}}$, we use as the discrepancy metric their *total variation*, defined by:

$$|\hat{\mathbf{p}} - \mathbf{p}| = \mathrm{d}_{\mathrm{var}}(\hat{\mathbf{p}}, \mathbf{p}) \stackrel{\mathrm{def}}{=} \frac{1}{2} \sum_{i=1}^{n} |\hat{p}_i - p_i| \,. \tag{5.3}$$

Theorem 10 *With an appropriate parameter m, the* dice-learning *algorithm (Algorithm 15) can accurately learn a multinomial distribution over n outcomes with a KWIK bound of*

$$B_{\textit{dice-learning}}(\epsilon, \delta) = m = \frac{2n}{\epsilon^2} \ln \frac{2n}{\delta} = \mathcal{O}\left(\frac{n}{\epsilon^2} \ln \frac{n}{\delta}\right).$$

Algorithm 15 Dice-learning.

Inputs: $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, \epsilon, \delta$.
Initialization:

$$c \leftarrow 0, \quad \hat{\mathbf{p}} \leftarrow \mathbf{0} = (0, 0, \ldots, 0) \in \mathbb{R}^n, \quad m \leftarrow \frac{2n}{\epsilon^2} \ln \frac{2n}{\delta}.$$

for $t = 1, 2, \ldots$ **do**
  Observe $\mathbf{x}_t \in \mathcal{X}$.
  if $c \leq m$ **then**
    Predict $\hat{\mathbf{y}}_t = \perp$ and observe $\mathbf{z}_t$.
    $c \leftarrow c + 1$.
    $\hat{\mathbf{p}} \leftarrow \hat{\mathbf{p}} + \frac{\mathbf{z}_t - \hat{\mathbf{p}}}{c}$.
  else
    Predict $\hat{\mathbf{y}}_t = \hat{\mathbf{p}}$.
  end if
end for

For comparison, note that KWIK-learning a multinomial distribution can be reduced to KWIK-learning a binomial distribution. We may *re-use* coin-learning to KWIK-learn each probability in the multinomial distribution $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ to ensure $2\epsilon/n$ accuracy with probability at least $1 - \delta/n$. Applying a union bound (Lemma 51), we conclude that with probability at least $1 - \delta$, every p_i is learned to within $\frac{2\epsilon}{n}$-accuracy, implying the total variation is at most ϵ:

$$|\hat{\mathbf{p}} - \mathbf{p}| = \frac{1}{2}\sum_{i=1}^{n} |\hat{p}_i - p_i| \leq \frac{1}{2}\sum_{i=1}^{n} \frac{2\epsilon}{n} = \epsilon,$$

The resulting KWIK bound for dice-learning is

$$B_{\text{coin-learning}}\left(\frac{2\epsilon}{n}, \frac{\delta}{n}\right) = \frac{n^2}{8\epsilon^2} \ln \frac{2n}{\delta} = \mathcal{O}\left(\frac{n^2}{\epsilon^2} \ln \frac{n}{\delta}\right),$$

which, however, is asymptotically worse by a factor of n than the stated bound.

However, a different analysis using a multiplicative form of the Chernoff [1952] inequality (Lemma 53) proves the KWIK bound given by Theorem 10 suffices to guarantee ϵ total variation [Kakade 2003, Lemma 8.5.5]. Although $B_{\text{dice-learning}}$ is asymptotically better than $B_{\text{coin-learning}}$, it is worse for small values of n due to the larger constant, which is a by-product of the more complicated analysis. Interested readers are also referred to Weissman et al. [2003] for a similar result.

Similar to coin-learning, the analysis and KWIK bound for dice-learning also applies to a slightly more general situation, where the hypothesis space is

$$\mathcal{H} = \{\mathbf{p} \in \mathbb{R}_+^n \mid \|\mathbf{p}\|_1 \leq C\}.$$

for some constant $C \in \mathbb{R}_+$. By re-scaling and introducing a dummy dimension, we may re-use the KWIK bound of dice-learning to obtain the following KWIK bound:

$$\mathcal{O}\left(\frac{nC^2}{\epsilon^2} \ln \frac{n}{\delta}\right).$$

The dice-learning algorithm will serve as an important building block in some of our KWIK applications to reinforcement learning in §7. Two other important stochastic problems are studied in the next subsections.

5.3.3 Learning the Mean of A Univariate Normal Distribution

An important continuous distribution is the univariate normal distribution, $\mathcal{N}(\mu, \sigma^2)$, whose probability density function and cumulative distribution function are given by

$$\varphi(x) \stackrel{\text{def}}{=} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

and

$$\Phi(x) \stackrel{\text{def}}{=} \Pr\left(y \leq x \mid y \sim \mathcal{N}(\epsilon, \sigma^2)\right) = \int_{-\infty}^{x} \varphi(y)\mathrm{d}y,$$

respectively, where $\mu \in \mathbb{R}$ is the mean and $\sigma^2 \in \mathbb{R}_+$ is the variance. For simplicity, this section considers the simplified problem of KWIK-learning the value of μ when the variance σ^2 is known *a priori*. The general problem of KWIK-learning *multivariate* normal distributions with *unknown* mean and covariance is considered in §6.6.

The algorithm is identical to coin-learning, and thus we omit the pseudocode descriptions. The only difference lies in the analytic technique used to quantify the KWIK bound, namely, the parameter m. In coin-learning, we may use Hoeffding's inequality as all observations are bounded random variables. But, this bound does not apply to a normally distributed random variable simply because it is unbounded. Below we discuss three approaches for determining m.

Using Hoeffding's Inequality

Two *indirect techniques* can be used to handle such unbounded random variables. One is to discard a random variable when its magnitude is large, and then argue that when m is large enough, with high probability, the sample set contains enough samples with small magnitude. This approach is adopted by Brunskill et al. [2008] to learn the mean vector of a multivariate normal distribution. Another way is to use a truncated normal distribution. Let $\tau \in \mathbb{R}_+$ be a constant, and $x \sim \mathcal{N}(\mu, \sigma^2)$. Then the induced random variable, $\bar{x}$, defined by

$$\bar{x} \stackrel{\text{def}}{=} \operatorname{sgn}(x) \min\{|x|, \tau\},$$

follows the τ-truncated normal distribution. In other words, we truncate the random variable x so that it is always within the range $[-\tau, \tau]$. If $|\mu| < 1$, it can be shown

that [Strehl and Littman 2008b; Li et al. 2009b]

$$|\mu - \mathbf{E}[\bar{x}]| \leq \frac{\epsilon}{2},$$

when

$$\tau \geq 1 + 2\sigma \ln \frac{16}{\sqrt{2\pi}\epsilon} = \mathcal{O}\left(\sigma \ln \frac{1}{\epsilon}\right).$$

Now, we may KWIK-learn $\mathbf{E}[\bar{x}]$ using coin-learning as $|\bar{x}| \leq \tau$. Applying Lemma 52, we have that the following sample size (or KWIK bound) guarantees that the maximum-likelihood estimate, $\hat{\mu}$, approximates $\mathbf{E}[\bar{x}]$ to within precision $\epsilon/2$ with probability $1-\delta$:

$$m = \frac{8\tau^2}{\epsilon^2} \ln \frac{1}{\delta} = \mathcal{O}\left(\frac{\sigma^2}{\epsilon^2} \ln^2 \frac{1}{\epsilon} \ln \frac{1}{\delta}\right). \tag{5.4}$$

And, we can easily show that $\hat{\mu}$ is an ϵ-accurate estimate of μ by the triangle inequality:

$$|\hat{\mu} - \mu| \leq |\hat{\mu} - \mathbf{E}[\bar{x}]| + |\mathbf{E}[\bar{x}] - \mu| \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon,$$

with probability at least $1 - \delta$.

Using Chebyshev's Inequality

A direct and simple way to get around unbounded random variables is to use Chebyshev's inequality (Lemma 54), which requires boundedness of the variance of the random variable, instead of the variable itself. This approach enables a simple analysis to decide a KWIK bound for learning the mean of a univariate normal distribution when the variance is known. We let the right-hand side of Chebyshev's inequality be δ and solve for m:

$$m = \frac{\sigma^2}{\delta\epsilon^2}. \tag{5.5}$$

Unfortunately, the dependence on $1/\delta$ is not logarithmic, because the tail bound in Chebyshev's inequality is not exponential. However, Equation 5.5 has a slightly better dependence on $1/\epsilon$ than Equation 5.4.

The result above can be generalized to a multivariate normal distribution, denoted $\mathcal{N}(\mu, \Sigma)$, in several ways. One way is to use one of the general techniques to combine individual KWIK learners that we will discuss in the next chapter (*c.f.*, §6.6, and in

particular, Lemma 17). Another way is to use the multidimensional form of Chebyshev's inequality (Lemma 57). A slightly improved result is recently proved by Monhor [2007] showing that the following sample size suffices to guarantee the maximum-likelihood estimate, $\hat{\mu}$, is ϵ-close to μ in ℓ_2-norm:

$$m = \mathcal{O}\left(\frac{\mathrm{tr}\,(\Sigma)}{\epsilon^2\delta}\right), \tag{5.6}$$

where the *trace*, $\mathrm{tr}\,(\Sigma)$, is the sum of the diagonal elements of Σ. Clearly, this KWIK bound is precisely a generalization of Equation 5.5.

Using Bernstein's Inequality

Bernstein's inequality (see Lemma 55 in the appendix for a precise statement) provides another way to analyze the KWIK bound of learning the mean of a univariate normal distribution. Like Hoeffding's inequality, it gives the desired exponential tail bounds; like Chebyshev's inequality, it allows one to reason with the original normal distribution directly without tricks such as truncation.

The key to using Lemma 55 is to find a constant c that satisfies the condition of this lemma (in particular, Equation B.1 in the appendix). In our problem of learning the mean of a normal distribution, we have the following technical lemma, whose proof is given in §5.6.1:

Lemma 12 *Let $x \sim \mathcal{N}(\mu, \sigma^2)$, then for all $k > 3$,*

$$\mathbf{E}\left[|x-\mu|^k\right] \leq (\sqrt{2}\sigma)^k\sqrt{k!}.$$

Lemma 12 implies immediately that Equation B.1 holds for univariate normal distributions with constant $c = 4\sigma/\sqrt{3} = \Theta(\sigma)$. We then let the right-hand side of Bernstein's inequality be δ and solve for m, resulting in the following KWIK bound:

$$m = \frac{\sigma^2 + \frac{4}{\sqrt{3}}\sigma\epsilon}{\epsilon^2}\ln\frac{1}{\delta} = \mathcal{O}\left(\frac{\sigma^2}{\epsilon^2}\ln\frac{1}{\delta}\right). \tag{5.7}$$

Clearly, this KWIK bound is asymptotically better than Equations 5.4 and 5.5. The analysis above proves the main theorem of this section.

Theorem 11 *The coin-learning algorithm can be used to KWIK-learn the mean of a univariate normal distribution with known variance, when m is set by Equation 5.7. The resulting KWIK bound is*

$$B_{univariate\ normal}(\epsilon, \delta) = \left(\frac{\sigma^2}{\epsilon^2} \ln \frac{1}{\delta}\right).$$

5.3.4 Subinterval Prediction

This section considers another way to generalize the learning problem in §5.3.1 that will be useful in studying a model-free reinforcement-learning algorithm in §8. In this problem, the learner is going to predict a subinterval of an interval that grows over time, but it only has access to noisy observations of numbers chosen arbitrarily from this interval.

Formally, we are given two known constants $L < U$, $\mathcal{X} = \mathbb{N}$, $\mathcal{Y} = \{(l, u) \mid L \leq l \leq u \leq U\} \subset \mathbb{R}^2$, $\mathcal{Z} = [L, U] \subset \mathbb{R}$, and $\mathcal{H} = \mathcal{Y}^{\mathcal{X}}$. Here, $\mathcal{Y}$ is viewed a set of closed intervals, so each element $y = (l, u) \in \mathcal{Y}$ is understood to be the closed interval $[l, u]$. At timestep t, the input is $x_t = t$, and the t-th output is $y_t = (l_t, u_t)$. The environment chooses a number $c_t \in y_t$ in an arbitrary way, and provides a stochastic observation, z_t, so that $\mathbf{E}[z_t] = c_t$. In addition, we assume that $l_1 \geq l_2 \geq l_3 \geq \cdots$, and $u_1 \leq u_2 \leq u_3 \leq \cdots$; namely, the output sequence, $[y_t]_{t \in \mathbb{N}}$, is a sequence of non-shrinking intervals with non-increasing lower endpoints and non-decreasing upper endpoints. The goal of the learner at timestep t is to predict $\perp$ or an interval, $\hat{y}_t = [\hat{l}_t, \hat{u}_t]$, such that $\hat{y}_t$ is a subinterval of y_t, modulo ϵ-error. Formally, we use the following discrepancy function:

$$|\hat{y}_t - y_t| = \max\left\{[l_t - \hat{l}_t]_+, [\hat{u}_t - u_t]_+\right\}, \tag{5.8}$$

where $[v]_+ \stackrel{\text{def}}{=} \max\{v, 0\}$ for all $v \in \mathbb{R}$. In other words, a predicted subinterval is not ϵ-accurate if its lower endpoint is too small or its upper endpoint is too large.

For the specific version of subinterval prediction problem defined above, we note two important observations. First, if $|\hat{y}_t - y_t| < \epsilon$, then $|\hat{y}'_t - y_t| < \epsilon$, where

$$\hat{y}'_t = \left(\frac{\hat{l}_t + \hat{u}_t}{2}, \frac{\hat{l}_t + \hat{u}_t}{2}\right)$$

is a degenerate interval. Therefore, we may assume that every non-$\perp$ prediction $\hat{y}_t$ is degenerate. Second, if some non-$\perp$ prediction $\hat{y}_t$ is made at timestep t, then the same prediction will be ϵ-correct for all future timesteps $\tau > t$, since the target intervals are non-shrinking: $y_1 \subseteq y_2 \subseteq y_3 \subseteq \cdots$.

To see the connection between subinterval prediction and coin-learning, consider the degenerate case where we assume $l_1 = u_1 = l_2 = u_2 = l_3 = u_3 = \cdots = c^*$ for some unknown constant c^*, and so each interval y_t becomes a singleton, $\{c^*\}$. Then, the discrepancy function of Equation 5.8 is equivalent to the requirements that the predictions be ϵ-close to c^*.

Perhaps unsurprisingly, the coin-learning algorithm can be used almost without modification to solve the subinterval prediction problem. The only difference in the algorithm is that, when making a prediction at timestep t, instead of predicting $\hat{y}_t = \hat{p}$, we need to predict $\hat{y}_t = (\hat{p}, \hat{p})$ since every prediction must be an interval in $\mathcal{Y}$. The main theorem of this section is similar to that for coin-learning:

Theorem 12 *The coin-learning algorithm can be used to solve the subinterval prediction problem, when m is set appropriately. The resulting KWIK bound is*

$$B_{subinterval}(\epsilon, \delta) = \left(\frac{(U-L)^2}{\epsilon^2} \ln \frac{1}{\delta} \right).$$

Proof. The proof is an application of Hoeffding's inequality. Suppose the algorithm has outputted m $\perp$s and observed the first m observations, $z_1, z_2, \ldots, z_m$. Define

$$p_m \stackrel{\text{def}}{=} \frac{c_1 + c_2 + \cdots + c_m}{m}.$$

Then, Hoeffding's inequality implies

$$\Pr\left(\hat{p} - p_m \geq \epsilon\right) \leq \exp\left(-\frac{2m\epsilon^2}{(U-L)^2}\right), \tag{5.9}$$

$$\Pr\left(\hat{p} - p_m \leq -\epsilon\right) \leq \exp\left(-\frac{2m\epsilon^2}{(U-L)^2}\right). \tag{5.10}$$

By setting the right-hand sides above to $\delta/2$ and solving for m, we have

$$m = \frac{(U-L)^2}{2\epsilon^2} \ln \frac{2}{\delta},$$

which guarantees the validity of Equations 5.9 and 5.10 with probability at least $1-\delta$. Therefore, with probability at least $1-\delta$, we have

$$l_t - \hat{p} < l_t - (p_m - \epsilon) = \frac{1}{m}\sum_{i=1}^{m}(l_t - c_i) + \epsilon = \frac{1}{m}\sum_{i=1}^{m}(l_i - c_i) + \epsilon \leq \epsilon$$

and

$$\hat{p} - u_t < (p_m + \epsilon) - u_t = \frac{1}{m}\sum_{i=1}^{m}(c_i - u_t) + \epsilon \leq \frac{1}{m}\sum_{i=1}^{m}(c_i - u_i) + \epsilon \leq \epsilon$$

for all $t > m$, which implies

$$|(\hat{p}, \hat{p}) - y_t| < \epsilon.$$

□

5.3.5 Stochastic Linear Regression

This section extends deterministic linear-regression (Algorithm 13) to the noisy case where the observations are target outputs corrupted by additive, white noise [Strehl and Littman 2008b]. In this setting, certain regularity assumptions are necessary in the problem formulation. Define $\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\|_2 \leq 1\}$, $\mathcal{Y} = \mathcal{Z} = [-1, 1]$, and

$$\mathcal{H} = \{f_{\mathbf{w}} \mid f_{\mathbf{w}}(x) = \mathbf{w}^\top \mathbf{x}, \mathbf{w} \in \mathbb{R}^n, \|\mathbf{w}\|_2 \leq 1\}.$$

That is, $\mathcal{H}$ is a set of linear functions on n variables. The target output is corrupted by additive, white noise: $z_t = y_t + \eta_t$, where η_t is a real-valued random variable with zero expectation. Note also that η_t is bounded since both $y_t \in \mathcal{Y}$ and $z_t \in \mathcal{Z}$ are bounded. Here, we describe the noisy linear-regression algorithm proposed by Strehl and Littman [2008b], which has the KWIK bound of:

$$B_{\text{noisy linear-regression}} = \tilde{\mathcal{O}}\left(\frac{n^3}{\epsilon^4}\right).$$

We note that an algorithm was recently developed by Walsh et al. [2009b], which enjoys a significantly improved KWIK bound of $\tilde{\mathcal{O}}\left(n/\epsilon^4\right)$.

In contrast, the deterministic case in §5.2.3 enjoys a significantly smaller KWIK bound in terms of n (linear instead of cubic). Here, the algorithm must be cautious to average over the noisy samples to make predictions accurately. Each time the algorithm returns $\perp$, it acquires a training input–output pair. Intuitively speaking, the algorithm

reports $\perp$ whenever it finds the current input does not lie in a subspace spanned by sufficiently many inputs in the training set. First, if the input does not lie close to the subspace spanned by inputs in the training data, no reliable prediction is possible, as in deterministic linear-regression. However, even if inputs in the training set span the whole input space, we still have to require a large enough number of training data to average out noise in the outputs.

Let $X \in \mathbb{R}^{m \times n}$ denote an $m \times n$ design matrix whose rows we interpret as transposed input vectors. We let $X(i)$ denote the transpose of the i-th row of X. Let $\mathbf{z} \in \mathbb{R}^m$ denote an m-dimensional vector whose i-th component, denoted $\mathbf{z}(i)$, is interpreted as the corresponding noisy observation.

Since $X^\top X$ is symmetric and positive semi-definite, it can be written as the following form of *singular value decomposition* [Golub and Van Loan 1996]:

$$X^\top X = U \Lambda U^\top, \tag{5.11}$$

where $U = [\mathbf{u}_1, \ldots, \mathbf{u}_n] \in \mathbb{R}^{n \times n}$, with $\mathbf{u}_1, \ldots, \mathbf{u}_n$ being a set of orthonormal singular vectors of $X^\top X$, and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, with corresponding singular values $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \geq 1 > \lambda_{k+1} \geq \cdots \geq \lambda_n \geq 0$. Note that Λ is diagonal but not necessarily invertible. Now, define $\bar{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_k] \in \mathbb{R}^{n \times k}$ and $\bar{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^{k \times k}$. For a fixed input $\mathbf{x}_t$ (a new input provided to the algorithm at time t), define

$$\bar{\mathbf{q}} \stackrel{\text{def}}{=} X \bar{U} \bar{\Lambda}^{-1} \bar{U}^\top \mathbf{x}_t \in \mathbb{R}^m, \tag{5.12}$$

$$\bar{\mathbf{u}} \stackrel{\text{def}}{=} \left[0, \ldots, 0, \mathbf{u}_{k+1}^\top \mathbf{x}_t, \ldots, \mathbf{u}_n^\top \mathbf{x}_t\right]^\top \in \mathbb{R}^n. \tag{5.13}$$

The main result about this algorithm is the following theorem. A proof sketch is provided due to Strehl and Littman [2008b] and detailed by Li et al. [2009b].

Theorem 13 *With appropriate parameter settings,* noisy linear-regression *is an efficient KWIK algorithm with a KWIK bound of* $\tilde{\mathcal{O}}\left(n^3/\epsilon^4\right)$.

Algorithm 16 Noisy linear-regression.

Inputs: $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, \epsilon, \delta, \alpha_1, \alpha_2$
Initialize $X = [\,]$ and $\mathbf{z} = [\,]$.
for $t = 1, 2, 3, \cdots$ **do**
  Observe $\mathbf{x}_t \in \mathcal{X}$.
  Compute $\bar{\mathbf{q}}$ and $\bar{\mathbf{u}}$ using Equations 5.12 and 5.13.
  if $\|\bar{\mathbf{q}}\|_2 \leq \alpha_1$ and $\|\bar{\mathbf{u}}\|_2 \leq \alpha_2$ **then**
    Solve for $\hat{\mathbf{w}}$:

$$\hat{\mathbf{w}} \leftarrow \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^n : \|\mathbf{w}\|_2 \leq 1} \|X\mathbf{w} - \mathbf{z}\|_2^2 .$$

    Predict $\hat{y}_t = \hat{\mathbf{w}}^\top \mathbf{x}_t$.
  else
    Predict $\hat{y}_t = \perp$.
    Receive observation z_t.
    Append $\mathbf{x}_t^\top$ as a new row to the matrix X.
    Append z_t as a new element to the vector $\mathbf{z}$.
  end if
end for

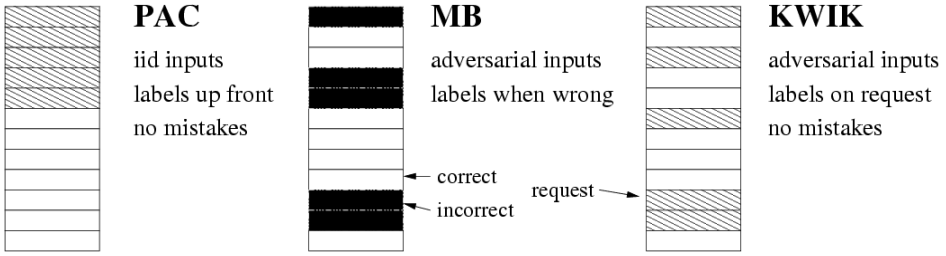


Figure 5.2: A Pictorial Comparison of PAC, MB, and KWIK (from Li et al. [2008]).

5.4 Related Learning Models

The KWIK model is most relevant to two existing learning models: PAC (probably approximately correct) [Valiant 1984] and MB (mistake bound) [Littlestone 1988]. Figure 5.2 gives a pictorial comparison among PAC, MB, and KWIK.

The PAC model is extensively studied mostly for *batch* learning a binary concept, where there is a separation between training and testing phases. In the training phase, a set of training input–output pairs, drawn IID from an unknown distribution, are provided to the learner, and the learner strives to infer the target function to within ϵ-accuracy in the testing phase. Precisely, let D be the distribution over the input set,

$\mathcal{X}$. PAC requires the inferred hypothesis, denoted $\hat{h}$, to be ϵ-accurate with probability at least $1-\delta$ with respect to the target function h^* in the following sense:

$$\mathbf{E}_{x\sim D}\left[\mathbb{I}(\hat{h}(x) \neq h^*(x))\right] \leq \epsilon.$$

Many hypothesis spaces are known to be learnable in the PAC model, such as the set of conjunctions and 3-CNF, but some are computationally intractable to learn under certain cryptographical assumptions [Kearns and Vazirani 1994].

The MB model is for *online* learning in which inputs may be chosen adversarially, both similar to KWIK. However, this model does not explicitly allow the option of predicting $\perp$ and so the learner must make a prediction $\hat{y}_t \in \mathcal{Y}$ in every timestep t, and some of these predictions may be wrong. To achieve successful learning in this model, the number of mistakes made by the learner has to be small (such as being polylogarithmic in the size of the hypothesis class). In binary classification, for example, the total number of mistakes may be defined as

$$\sum_{t=1}^{\infty} \mathbb{I}(\hat{y}_t \neq y_t).$$

The MB model is shown to be polynomially equivalent to the learning-by-query model [Angluin 1988]: if a hypothesis class $\mathcal{H}$ is learnable in the MB model, then it is also learnable in the query model, and vice versa.

Littlestone [1989] proves that a MB-learnable hypothesis space must be PAC-learnable. On the other hand, Blum [1994] shows that there exists a hypothesis space that is efficiently PAC-learnable but not efficiently MB-learnable, assuming the existence of one-way functions, a cryptographical conjecture that is widely assumed to be true (see, *e.g.*, Papadimitriou [1994]). Therefore, MB appears to be a strictly harder model than PAC. The following examples show that KWIK is strictly harder than MB.

Example 10 *A MB-learnable, but not KWIK-learnable example is Boolean conjunctions. Define* $\mathcal{X} = \mathbb{B}^n, \mathcal{Y} = \mathcal{Z} = \mathbb{B}$, *and*

$$\mathcal{H} = \{h \mid h(\mathbf{x}) = x_{i_1} \wedge x_{i_2} \wedge \cdots \wedge x_{i_k},\ \textit{where } 0 \leq k \leq n \textit{ and } i_1, \ldots, i_k \in \{1, 2, \ldots, n\}\}.$$

By convention, define $h(\mathbf{x}) \equiv 1$ *when no variable is used in the conjunction. Let*

observations be deterministic. We define $\dim(\mathcal{H}) = n$*, and so desire an algorithm with at most* $\mathrm{poly}(n)$ *mistakes in the MB model, or at most* $\mathrm{poly}(n)$ $\perp$*s in the KWIK model.*

This hypothesis class is MB-learnable with no more than n *mistakes by the following algorithm. It initializes a set* R *by* $\{1, 2, \ldots, n\}$*. For an input* $\mathbf{x}$*, if* $x_i = 1$ *for all* $i \in R$*, it predicts* $\hat{y} = 1$ *(*TRUE*), and otherwise,* $\hat{y} = 0$ *(*FALSE*). It then gets to observe the true output,* y*. If* $\hat{y} = 0$ *and* $\hat{y} = 1$*, it updates* R *by:* $R \leftarrow R \setminus \{i \mid x_i = 0\}$*. First, it can be observed that* R *is always a superset of the indices of the variables involved in the target conjunction,* h^**. Therefore, a "false positive" error (namely,* $\hat{y} = 1$ *and* $y = 0$*) is impossible, and so the number of mistakes made by the algorithm is upper bounded by the number of "false negative" errors (namely,* $\hat{y} = 0$ *and* $y = 1$*). Second, we can observe that every false negative error reduces the size of* R *by at least one. Since* $|R| = n$ *initially and is non-negative, the total number of mistakes is at most* n*.*

We now show that it may require $2^n - 1$ *many* $\perp$*s to learn* $\mathcal{H}$ *in the KWIK model. Let the target function be the conjunction of* all *variables; namely,* $h^*(\mathbf{x}) = x_1 \wedge x_2 \wedge \cdots \wedge x_n$*. We now construct a worst-case sequence of inputs to get the* $2^n - 1$ *KWIK bound:* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{2^n-1}$*. These inputs are distinct and are organized in* n *groups:*

- *The first group consists of the first input* $\mathbf{x}_1 = [0, 0, \ldots, 0]^\top$*;*
- *The second group consists of the next* n *inputs, each of which has exactly one* 1 *in one component and* 0 *elsewhere; namely,* $\mathbf{x}_t = [0, 0, \ldots, 0, 1, 0, \cdots, 0]^\top$*, in which the* 1 *appears in the* $(t-1)$*-st position for* $2 \le t \le n + 1$*;*
- *The third group consists of the next* $\binom{n}{2} = n(n-1)/2$ *inputs, each of which has exactly two* 1*s in two components and* 0 *elsewhere;*
- ...
- *The last group consists of the final* $\binom{n}{n-1} = n$ *inputs, each of which has exactly* $n - 1$ 1*s in* $n - 1$ *components and* 0 *in the rest.*

The total number of inputs above is

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n-1} = 2^n - 1.$$

The correct output for all labels above are FALSE*, but at any timestep* $t < 2^n$*, the input–output observations up to* $t - 1$ *do not reveal any positive instance of* h^**. The*

way we construct the inputs guarantees that in all these timesteps, the correct output can be either TRUE *or* FALSE*, without knowing* h^**. Only until* y_{2^n-1} = FALSE *is observed can the learner realize that* h^* *is the conjunction of all* n *variables and start making accurate predictions.*

In addition to PAC and MB, KWIK also shares elements with a few other learning models. First, we consider a generalization of the number of mistakes known as *loss*, which applies to both classification and regression problems. The learning process is similar to that in MB: we also denote the input, prediction, and output at timestep t by x_t, $\hat{y}_t$, and y_t, respectively. The learner is informed of a *loss function* $\ell : \mathcal{Y}^2 \to \mathbb{R}_+$; the function value, $\ell(\hat{y}_t, y_t)$, is understood to be a measure of how inconsistent or different the prediction $\hat{y}_t$ is from the ground truth y_t. For example, in binary classification problems the 0/1-loss function is often used:

$$\ell(\hat{y}_t, y_t) = \mathbb{I}(\hat{y}_t \neq y_t);$$

whereas in a regression problem where $\mathcal{Y} \subseteq \mathbb{R}$, ℓ may be the square loss function:

$$\ell(\hat{y}_t, y_t) = (\hat{y}_t - y_t)^2 .$$

The total H-step loss of the learner is

$$\sum_{t=1}^{H} \ell(\hat{y}_t, y_t),$$

which may be infinite as $H \to \infty$. We can now define *H-step regret* $L(H)$ as the difference between the total loss of the learner and the total loss of the best hypothesis of the hypothesis space in hindsight:

$$L(H) \stackrel{\text{def}}{=} \sum_{t=1}^{H} \ell(\hat{y}_t, y_t) - \min_{h \in \mathcal{H}} \left\{ \sum_{t=1}^{H} \ell(h(x_t), y_t) \right\}.$$

If $L(H) = o(H)$, we know the learner's total loss will approach that of the best hypothesis in hindsight, and thus call the learner (or the learning algorithm) *no-regret*. See §4.3.1 for a similar definition of regret.

Another model similar to MB is the *regret* framework as applied to *associative bandit problems* [Strehl et al. 2006d]: at every timestep t in this model, the learner receives

input $x_t \in \mathcal{X}$, selects an action a_t from a possibly infinite set $\mathcal{A}$, and then receives a randomized payoff $r_t \in \mathbb{R}$, whose expectation depends on x_t and a_t. The goal is to minimize the regret, defined as the difference between the largest total payoff by following the best action-selection rule π in some given rule set Π and the total payoff received by the learner. *No-regret algorithms* have been found for variants of associative bandit problems in the sense that their regrets are sublinear in the number of timesteps (*e.g.*, Auer [2002]). Consequently, the average per-timestep regret converges to 0 in the limit. Under certain conditions and transformations, we can equate regret with the number of prediction mistakes, and thus the overall probability of making a mistake of a no-regret algorithm converges to 0. However, these algorithms, like MB algorithms, may make inaccurate predictions, violating the accuracy requirement of the KWIK model.

Conformal prediction [Shafer and Vovk 2008] is an online-learning paradigm in which the learner has to make a *region*, rather than a *point*, prediction, for the present input based on previously observed input–output pairs. It is required that these prediction regions contain the correct output with high probability. It is straightforward to decide whether the output is known within sufficient accuracy based on the "size" of the region. For example, in regression problems, if the region is an interval of length smaller than ϵ, then any point prediction in this region will be ϵ-accurate with high probability. However, existing conformal-prediction methods make statistical assumptions about inputs such as independence or exchangeability, and thus are rendered inapplicable in the KWIK setting.

The notion of allowing the learner to opt out of some inputs by returning $\perp$ is not unique to KWIK. Several other authors have considered related models. For example:

- In the MB-like *apple-tasting* setting [Helmbold et al. 2000], the learner receives feedback asymmetrically only when it predicts a particular label (a positive example, say), which conflates the request for a sample with the prediction of a particular outcome.
- *Sleeping experts* [Freund et al. 1997a] can respond $\perp$ for some inputs, although they need not learn from these experiences and the number of $\perp$s in the whole

run may be unbounded.

- Learners in the settings of *selective sampling* [Cesa-Bianchi et al. 2006] and *label efficient prediction* [Cesa-Bianchi et al. 2005] request labels randomly with a changing probability and achieve bounds on the expected number of mistakes and the expected number of label requests for a finite number of interactions. These algorithms cannot be used unmodified in the KWIK setting because, with high probability, KWIK algorithms must not make mistakes at any time.
- *Query-by-committee* algorithms [Seung et al. 1992; Freund et al. 1997b] determines the confidence level of its prediction for an example based on the degree of disagreements in the predictions of sub-algorithms in the committee. Similarly to selective sampling algorithms, query-by-committee algorithms were proposed for active learning and may not possess the accuracy requirement as KWIK.
- Finally, the averaged classifier by Freund et al. [2004] may return $\perp$ if the averaged prediction for an example is close to the classification boundary. However, it is assumed that examples are *i.i.d.* and a non-$\perp$ prediction may not be accurate.

5.5 Further Discussions

So far in this chapter, we have defined KWIK and investigated some of its basic properties under the realization assumption. In this section, we briefly discuss a few potential extensions.

5.5.1 Agnostic KWIK Learning

For simplicity, the dissertation focuses on the *realizable* case where the hypothesis class must contain the true hypothesis: $h^* \in \mathcal{H}$. A natural extension is the *agnostic* case that removes such an assumption. This subsection gives a preliminary study of agnostic KWIK learning. We first show a simple case where agnostic KWIK learning is possible, then give a hypothesis class that cannot be KWIK-learned agnostically, and finally provide a general impossibility result by a reduction to online learning with malicious noise.

Before going into positive and negative examples, it is necessary to define when a learning algorithm is deemed successful in the agnostic setting. It is possible that *no* hypothesis in the hypothesis class is able to make accurate predictions for all inputs. Motivated by the agnostic PAC model [Kearns et al. 1994], a natural way to define the problem is to compare the prediction error of the learner to the prediction error of the best hypothesis in $\mathcal{H}$. Specifically, we may replace the accuracy requirement in Definition 11 by: "There is a constant $c \geq 1$ so that if $\hat{y}_t \neq \perp$, it must be close to the best hypothesis' prediction: $|\hat{y}_t - y_t| < \epsilon + c\xi$", where

$$\xi = \inf_{h \in \mathcal{H}} \|h - h^*\|_\infty = \inf_{h \in \mathcal{H}} \sup_{x \in \mathcal{X}} |h(x) - h^*(x)|$$

is the smallest ℓ_∞-error of the best hypothesis in $\mathcal{H}$, and h^* is the target function and may not be in $\mathcal{H}$. Note that the realization assumption guarantees $\xi = 0$ as $h^* \in \mathcal{H}$.

The following algorithm which we call agnostic enumeration (Algorithm 17) is due to Michael Littman[2] and can be viewed as a robust version of enumeration in the agnostic and deterministic case when $\mathcal{H}$ is finite and $\mathcal{Y} \subset \mathbb{R}$. It can KWIK-learn $\mathcal{H}$ when $c \geq 2$ with a KWIK bound of

$$B_{\text{agnostic enumeration}} = |\mathcal{H}| - 1.$$

Algorithm 17 Agnostic enumeration.

Inputs: $\mathcal{X}, \mathcal{Y}, \mathcal{H}, \epsilon, \delta, \xi$.
$\hat{\mathcal{H}} \leftarrow \mathcal{H}$.
for $t = 1, 2, \ldots$ **do**
  Observe $x_t \in \mathcal{X}$.
  Compute $\hat{L} = \{h(x_t) \mid h \in \hat{\mathcal{H}}\}$.
  $l_{\max} \leftarrow \max_{l \in \hat{L}}$ and $l_{\min} \leftarrow \min_{l \in \hat{L}}$.
  if $l_{\max} - l_{\min} < 2\xi$ **then**
    Predict $\hat{y}_t = (l_{\max} + l_{\min})/2$.
  else
    Predict $\hat{y}_t = \perp$ and observe y_t.
    $\hat{\mathcal{H}} \leftarrow \hat{\mathcal{H}} \setminus \{h \mid h \in \hat{\mathcal{H}} \wedge |h(x_t) - y_t| \geq \xi\}$.
  end if
end for

In general, however, agnostic KWIK learning may not be possible for small values of c, as shown by the following example of learning a linear function:

[2]Personal communications, 2008.

Example 11 *Let $\mathcal{X} = \mathcal{Y} = \mathcal{Z} = [0,1]$ and require $c = 1$. The observations are deterministic. Let $\epsilon < \xi$. The target function f to learn is the zero function (which is trivially linear): $f(x) \equiv 0$. Let the input at time t be $x_t = \min\{t\beta, 1\}$ for some small $\beta > 0$, then the corresponding output is $y_t = f(x_t) = 0$. Assume that the learner knows f is noise free. But, since it does not know that f is exactly linear, it has to predict conservatively to handle the worst-case situations. At time t where $\frac{\xi}{\beta} < t < \frac{1}{\beta}$, the learner has to predict $\perp$, based on the training data up to time $t-1$. The possible range of y_t, which is $[-\frac{2t\xi}{t-1}, \frac{2t\xi}{t-1}]$, is too wide to guarantee a prediction error of at most $\xi + \epsilon$. By letting $\beta \downarrow 0$, the number of $\perp$s does not depend on ϵ or ξ, and is unbounded.*

Finally, we note that since KWIK-learning is strictly harder than PAC learning with malicious noise [Kearns and Li 1993], their bounds for maximum possible error rates that can be tolerated by any learning algorithm applies directly to the KWIK model.

5.5.2 Dimension in KWIK Learning

An open problem in KWIK is how we may characterize the dimension or complexity of a hypothesis class, which would be useful in characterizing the sample-complexity bounds, similar to the VC dimension. A natural idea is to define a combinatorial dimension that helps in this direction. For simplicity, we only consider the noise-free and realizable case in this section. The following definition is motivated by a similar combinatorial parameter proposed by Kearns and Li [1993].

Definition 12 *A hypothesis class H is* s-splittable with precision ϵ *if there exist s input–output pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ and s distinct hypotheses $h_1, ..., h_s \in \mathcal{H}$ such that:*

1. *$x_i \neq x_j$ if $i \neq j$;*
2. *$|h_i(x_i) - y_i| \geq \epsilon$, but $|h_j(x_i) - y_i| < \epsilon$ for all other j.*

In other words, the i-th data can only rule out hypothesis h_i, but not other h_j. The complexity metric, denoted $\dim_s(\mathcal{H}, \epsilon)$*, is then defined as the largest possible s.*

It is useful to compare $\dim_s$ to other complexity measures in computational learning theory. To allow a reasonable comparison, we assume that $\mathcal{Y} = \mathbb{B}$ and so $\mathcal{H}$ contains

binary concepts. First, it can be shown that $\dim_s(\mathcal{H})$ is never smaller than the *VC-dimension* of $\mathcal{H}$ [Vapnik 2006]. Second, $\dim_s(\mathcal{H})$ bears some similarity to the *teaching dimension* [Goldman and Kearns 1995], but is never smaller than the teaching dimension.

A few simple observations follow directly from definition. The proofs are omitted.

Lemma 13 *Let $B(\epsilon, \delta)$ be the KWIK bound of an algorithm* **A** *for a hypothesis class $\mathcal{H}$. If $\delta < 1$, then $B(\epsilon, \delta) \geq \dim_s(\mathcal{H}, \epsilon)$.*

Lemma 14 *There are problems where $\dim_s(\mathcal{H}) = |\mathcal{H}| - 1$. So enumeration is optimal in some cases.*

Lemma 15 *There are problems where $\dim_s(\mathcal{H}) = |\mathcal{X}|$. So memorization is optimal in some cases.*

Lemma 16 *In the deterministic linear regression problem defined in §5.2.3, $\dim_s(\mathcal{H}) = n$, implying optimality of the deterministic linear-regression algorithm.*

Unfortunately, $\dim_s(\mathcal{H})$ cannot be used to derive KWIK upper bounds, even if we ignore computational limitations. The following example shows that there is a hypothesis class $\mathcal{H}$ with a small $\dim_s(\mathcal{H})$ but cannot be KWIK-learned by any algorithm.

Example 12 *Let $\mathcal{X} = [0, 1]$, $\mathcal{Y} = \mathcal{Z} = \mathbb{B}$, and $\mathcal{H} = \{h \mid h(x) = \mathbb{I}(x \geq \theta), \theta \in [0, 1]\}$. So, the hypothesis class contains all step functions defined on $\mathcal{X}$. Observations are deterministic. Suppose the target function uses $\theta = 1$; that is, $h^*(x) = \mathbb{I}(x \geq 1)$. Furthermore, the sequence of inputs are $x_t = 1 - 1/t$. If $\epsilon < 1$, then any KWIK algorithm must predict $\perp$ for all inputs, and the KWIK bound is ∞. However, $\dim_s(\mathcal{H}) = 2$.*

5.6 Proofs

This section provides detailed proofs of all technical lemmas used in this chapter.

5.6.1 Proof of Lemma 12

PROOF. Let $x \sim \mathcal{N}(\mu, \sigma^2)$. For all $p \in \mathbb{N}$, Lemma 60 states that

$$\mathbf{E}\left[|x-\mu|^{2p}\right] = \frac{(2p)!\sigma^{2p}}{2^p p!}. \tag{5.14}$$

Notice that

$$\begin{aligned}
\frac{(2p)!}{2^p p!} &= \frac{(2p)\cdot(2p-1)\cdot(2p-2)\cdot(2p-3)\cdots 3\cdot 2\cdot 1}{(2p)\cdot(2(p-1))\cdot(2(p-2))\cdots(2\cdot 3)\cdot(2\cdot 2)\cdot(2\cdot 1)} \\
&\leq \frac{(2p)\cdot(2p)\cdot(2p-2)\cdot(2p-2)\cdots 3\cdot 2\cdot 1}{(2p)\cdot(2(p-1))\cdot(2(p-2))\cdots(2\cdot 3)\cdot(2\cdot 2)\cdot(2\cdot 1)} \\
&= (2p)\cdot(2(p-1))\cdot(2(p-2))\cdots(2\cdot 3)\cdot(2\cdot 2)\cdot(2\cdot 1) \\
&= 2^p p!.
\end{aligned}$$

Therefore,

$$2^p p! \geq \sqrt{(2p)!}, \tag{5.15}$$

implying that

$$\mathbf{E}\left[|x-\mu|^{2p}\right] = \frac{(2p)!\sigma^{2p}}{2^p p!} \leq 2^p p!\sigma^{2p} \leq \left(\sqrt{2}\sigma\right)^{2p}\sqrt{(2p)!}.$$

We have thus proved the lemma for even k.

For odd k, we will need the Cauchy-Schwarz inequality (Lemma 67). Specifically,

$$\mathbf{E}\left[|x-\mu|^{k}\right] \leq \sqrt{\mathbf{E}\left[|x-\mu|^{2k}\right]} = \sqrt{\frac{(2k)!\sigma^{2k}}{2^k k!}} \leq \sigma^k\sqrt{2^k k!},$$

where the first step applies Lemma 67 with $u = |x-\mu|^k$ and $v = 1$, the second uses Equation 5.14, and the last uses Equation 5.15. We have thus proved the lemma also for odd k, and the proof is complete. □

Chapter 6

Combining KWIK Learners

This section provides examples of how KWIK learners can be combined to provide learning guarantees for more complex hypothesis classes. In all cases, we are given a collection of k KWIK problems with respective input, output, observation, and hypothesis sets, together with corresponding KWIK algorithms for these problems. Each of these algorithms is referred to a *sub-algorithm*. The goal here is to create a *master algorithm*, which makes use of the sub-algorithms, for solving a new KWIK problem that is defined using the k KWIK problems in various ways. Applications of these master algorithms in PAC-MDP reinforcement learning are the topic of Part III.

6.1 Input Partition

We first consider a variant of memorization that combines learners across disjoint input sets. Let $\mathcal{X}_1, \ldots, \mathcal{X}_k$ be a set of disjoint input sets ($\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$ if $i \neq j$) and $\mathcal{Y}$ be an output set. Let $\mathcal{H}_1, \ldots, \mathcal{H}_k$ be a set of KWIK-learnable hypothesis classes with KWIK bounds of $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$ where $\mathcal{H}_i \subseteq \mathcal{Y}^{\mathcal{X}_i}$. The input-partition algorithm (Algorithm 18) can be used to KWIK-learn the hypothesis class

$$\mathcal{H} \stackrel{\text{def}}{=} \mathcal{H}_1 \times \cdots \times \mathcal{H}_k \subseteq \mathcal{Y}^{\mathcal{X}_1 \cup \cdots \cup \mathcal{X}_k}.$$

Note that input-partition applies both in deterministic and in stochastic problems.

Input-partition runs a sub-algorithm, denoted $\mathbf{A}_i$, for each subclass $\mathcal{H}_i$. When it receives an input $x_t \in \mathcal{X}_i$ at timestep t, it queries $\mathbf{A}_i$ and returns its response $\hat{y}_t$. If $\hat{y}_t = \perp$, an observation is obtained and input-partition informs $\mathbf{A}_i$ of this observation to allow it to learn. Otherwise, $\hat{y}_t$ is ϵ-accurate with high probability, since $\mathbf{A}_i$ is a KWIK algorithm for $\mathcal{H}_i$. The total number of $\perp$s is the sum of the number of $\perp$s returned by

Algorithm 18 Input-partition.

0: **Inputs:** $\mathcal{X}_1, \ldots, \mathcal{X}_k, \mathcal{Y}, \mathcal{Z}, \mathcal{H}_1, \ldots, \mathcal{H}_k, \epsilon, \delta, \mathbf{A}_1, \ldots, \mathbf{A}_k$.
1: Each sub-algorithm $\mathbf{A}_i$ is run using parameters ϵ and δ/k.
2: **for** $t = 1, 2, \ldots$ **do**
3: Observe $x_t \in \mathcal{X} \stackrel{\text{def}}{=} \mathcal{X}_1 \cup \cdots \cup \mathcal{X}_k$ and find i such that $x_t \in \mathcal{X}_i$.
4: Run $\mathbf{A}_i$ with input x_t and make $\hat{y}_t$ the return of $\mathbf{A}_i$ on x_t.
5: **if** $\hat{y}_t = \perp$ **then**
6: Observe z_t and feed it to $\mathbf{A}_i$ to allow it to learn.
7: **end if**
8: **end for**

all sub-algorithms $\mathbf{A}_i$. To achieve $1 - \delta$ certainty, it insists on $1 - \delta/k$ certainty from each of the sub-algorithms. By a union bound, the overall failure probability is at most the sum of the failure probabilities of the sub-algorithms, which is $k \cdot \delta/k = \delta$. We have thus proved the main result in this section.

Theorem 14 *Input-partition can KWIK-learn the hypothesis class, $\mathcal{H} \stackrel{\text{def}}{=} \mathcal{H}_1 \times \cdots \times \mathcal{H}_k$ with a KWIK bound of:*

$$B_{\textit{input-partition}}(\epsilon, \delta) = \sum_{i=1}^{k} B_i \left(\epsilon, \frac{\delta}{k} \right).$$

Example 13 *Let $\mathcal{X} = \mathcal{Y} = \mathbb{R}$. Define $\mathcal{H}$ to be a set of piecewise linear functions:*

$$\mathcal{H} = \{ f \mid f(x) = px \text{ if } x \geq 0, f(x) = mx \text{ otherwise}, p \in \mathbb{R}, m \in \mathbb{R} \}.$$

Using Algorithm 13, we can KWIK-learn the class of linear functions over two input sets, $\mathcal{X}_- = (-\infty, 0)$ and $\mathcal{X}_+ = [0, \infty)$, each requiring a KWIK bound of 1. Note that $\{\mathcal{X}_-, \mathcal{X}_+\}$ is a partition *of the entire input set: $\mathcal{X}_- \cup \mathcal{X}_+ = \mathcal{X}$ and $\mathcal{X}_- \cap \mathcal{X}_+ = \emptyset$. We can use Algorithm 18 to KWIK-learn $\mathcal{H}$ with a KWIK bound of $1 + 1 = 2$. The two KWIK learners are called $\mathbf{A}_-$ and $\mathbf{A}_+$, respectively.*

Assume the first input is $x_1 = 2$, which is in $\mathcal{X}_+$. Input-partition queries $\mathbf{A}_+$ with input x_1. Since $\mathbf{A}_+$ has no idea about y_1, it returns $\perp$. Hence, input-partition reports $\hat{y}_1 = \perp$, and $y_1 = 4$ is observed. Learner $\mathbf{A}_+$ can now infer with certainty that $p = y_1/x_1 = 2$, and we can now predict $f(x)$ for all $x \in \mathcal{X}_+$. The next input is $x_2 = 0.5 \in \mathcal{X}_+$, which is again presented to $\mathbf{A}_+$, resulting in $\hat{y}_2 = px_2 = 1$. The third input is $x_3 = -1 \in \mathcal{X}_-$. The algorithm queries $\mathbf{A}_-$ with input x_3 and receives $\perp$ since $\mathbf{A}_-$

does not have enough information to predict y_3. The algorithm then predicts $\perp$ for the second time and sees $y_3 = 3$. Learner $\mathbf{A}_-$ can now determine $m = y_3/x_3 = -3$, and the target function is completely identified:

$$f(x) = \begin{cases} 2x & \text{if } x \geq 0 \\ -3x & \text{otherwise.} \end{cases}$$

6.2 Output Combination

A similar approach applies to the output set $\mathcal{Y}$ when it is a cross product of k sets: $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_k$. Since the accuracy requirement in Definition 11 depends on the output set as well as the interpretation of the discrepancy metric $|\cdot|$, it is necessary to make certain assumptions to relate the discrepancy metric in $\mathcal{Y}$ to those in $\mathcal{Y}_i$. A natural choice, which we will use in §7, is to assume the existence of some $\alpha \in \mathbb{R}_+$ such that for any $\epsilon \in (0, 1)$ and any $y_i, \hat{y}_i \in \mathcal{Y}_i$, if $|\hat{y}_i - y_i| < \alpha\epsilon$ for all $i = 1, 2, \ldots, k$, then $|\hat{y} - y| < \epsilon$, where $y = (y_1, \ldots, y_k)$ and $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_k)$. For example, if we use the ℓ_1, ℓ_2, or ℓ_∞ norms in the output spaces $\mathcal{Y}$ and $\mathcal{Y}_i$, then α can be $1/k$, $1/\sqrt{k}$, and 1, respectively.

Let $\mathcal{X}$ be an input set and $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$ be a collection of output sets. Let $\mathcal{H}_1, \ldots, \mathcal{H}_k$ be a set of KWIK-learnable hypothesis classes with KWIK bounds of $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$ where $\mathcal{H}_i \subseteq \mathcal{Y}_i^{\mathcal{X}}$. The output-combination algorithm can be used to KWIK-learn the hypothesis class for the composite output set, $\mathcal{Y} \stackrel{\text{def}}{=} \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_k$:

$$\mathcal{H} \stackrel{\text{def}}{=} \mathcal{H}_1 \times \cdots \times \mathcal{H}_k \subseteq (\mathcal{Y}_1 \times \cdots \times \mathcal{Y}_k)^{\mathcal{X}}.$$

Note that ouptut-combination applies both in deterministic and in stochastic problems.

Output-combination works by running each sub-algorithm, $\mathbf{A}_i$, simultaneously with parameters $\alpha\epsilon$ and δ/k. Given an input $x_t \in \mathcal{X}$, it collects the returns of all sub-algorithms. If any one of them returns $\perp$, output-combination also returns $\perp$ to acquire an observation z_t to allow the sub-algorithms to learn. Otherwise, it simply combines the k component predictions and returns the assembled prediction $\hat{y}_t$. Since each component prediction is $\alpha\epsilon$-accurate, the combined prediction must be ϵ-accurate. Furthermore, the total number of $\perp$s returned by output-combination is the sum of $\perp$s returned

Algorithm 19 Output-combination.

Inputs: $\mathcal{X}, \mathcal{Y}_1, \ldots, \mathcal{Y}_k, \mathcal{Z}, \mathcal{H}_1, \ldots, \mathcal{H}_k, \epsilon, \delta, \mathbf{A}_1, \ldots, \mathbf{A}_k$.
Each sub-algorithm $\mathbf{A}_i$ is run using parameters $\alpha\epsilon$ and δ/k.
for $t = 1, 2, \ldots$ **do**
  Observe $x_t \in \mathcal{X}$.
  for all $i \in \{1, 2, \ldots, k\}$ **do**
    Run $\mathbf{A}_i$ with input x_t, and let $\hat{y}_t^i$ be the sub-algorithm's return.
  end for
  if $\hat{y}_t^i = \perp$ for any i **then**
    Predict $\hat{y}_t = \perp$ and observe z_t.
    Feed z_t to all $\mathbf{A}_i$ with $\hat{y}_t^i = \perp$ to allow them to learn.
  else
    Predict $\hat{y}_t = (\hat{y}_t^1, \hat{y}_t^2, \ldots, \hat{y}_t^k)$.
  end if
end for

by the sub-algorithms. Finally, a union bound implies the overall failure probability is at most the sum of the failure probabilities of the sub-algorithms, which is $k \cdot \delta/k = \delta$. We have thus proved the main result of this section.

Theorem 15 *Output-combination can KWIK-learn the hypothesis class* $\mathcal{H} \stackrel{\text{def}}{=} \mathcal{H}_1 \times \cdots \times \mathcal{H}_k$ *with a KWIK bound of*

$$B_{\text{output-combination}}(\epsilon, \delta) = \sum_{i=1}^{k} B_i\left(\alpha\epsilon, \frac{\delta}{k}\right).$$

6.3 Cross Product

The next algorithm generalizes the previous algorithm by combining both the input and output sets. Let $\mathcal{X}_1, \ldots, \mathcal{X}_k$ and $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$ be a set of input and output sets and $\mathcal{H}_1, \ldots, \mathcal{H}_k$ be a collection of KWIK-learnable hypothesis classes with KWIK bounds of $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$ on these sets. That is, $\mathcal{H}_i \subseteq \mathcal{Y}_i^{\mathcal{X}_i}$. Let $\mathcal{X} \stackrel{\text{def}}{=} \mathcal{X}_1 \times \cdots \times \mathcal{X}_k$ and $\mathcal{Y} \stackrel{\text{def}}{=} \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_k$ be the composite input and output sets, respectively. The cross-product algorithm (Algorithm 20) can learn the hypothesis class

$$\mathcal{H} \stackrel{\text{def}}{=} \mathcal{H}_1 \times \cdots \times \mathcal{H}_k \subseteq (\mathcal{Y}_1 \times \cdots \times \mathcal{Y}_k)^{\mathcal{X}_1 \times \cdots \times \mathcal{X}_k}.$$

Note that cross-product applies both in deterministic and stochastic problems.

Here, each input consists of a vector of inputs from each of the sets $\mathcal{X}_1, \ldots, \mathcal{X}_k$ and outputs are vectors of outputs from $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$. Like Algorithm 19, each component

Algorithm 20 Cross-product.

Inputs: $\mathcal{X}_1, \ldots, \mathcal{X}_k, \mathcal{Y}_1, \ldots, \mathcal{Y}_k, \mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{H}_1, \ldots, \mathcal{H}_k, \epsilon, \delta, \mathbf{A}_1, \ldots, \mathbf{A}_k$.
Each sub-algorithm $\mathbf{A}_i$ is run using parameters $\alpha\epsilon$ and δ/k.
for $t = 1, 2, \ldots$ **do**
  Observe $x_t = (x_t^1, \ldots, x_t^k) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_k$.
  Run $\mathbf{A}_i$ with input x_t^i for all i. Denote their returns by $\hat{y}_t^i$.
  if $\hat{y}_t^i = \perp$ for any i **then**
    Predict $\hat{y}_t = \perp$ and observe $z_t = (z_t^1, \ldots, z_t^k) \in \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_k$.
    Feed z_t^i to all $\mathbf{A}_i$ with $\hat{y}_t^i = \perp$ to allow them to learn.
  else
    Predict $\hat{y}_t = (\hat{y}_t^1, \hat{y}_t^2, \ldots, \hat{y}_t^k)$.
  end if
end for

of the output vector can be learned independently via the corresponding algorithm. Each is learned to within an accuracy of $\alpha\epsilon$ and confidence $1 - \delta/k$. Any time any component returns $\perp$, cross-product returns $\perp$. Since each $\perp$ returned can be traced to one of the sub-algorithms, the total is bounded as described above. By a union bound, total failure probability is no more than $k \times \delta/k = \delta$. We have thus proved the main result of this section.

Theorem 16 *Cross-product can KWIK-learn the composite hypothesis class* $\mathcal{H} \stackrel{\text{def}}{=} \mathcal{H}_1 \times \cdots \times \mathcal{H}_k$ *with a KWIK bound of*

$$B_{\text{cross-product}}(\epsilon, \delta) = \sum_i B_i \left(\alpha\epsilon, \frac{\delta}{k} \right).$$

We note that Algorithm 19 is a degenerate case of this algorithm where $\mathcal{X}_i = \mathcal{X}_j$ for all i, j and every input tuple $x_t = (x, \ldots, x)$ for some $x \in \mathcal{X}_1$.

6.4 Union

The previous two algorithms concern combinations of input or output sets and apply to both deterministic and noisy observations. We next provide an intuitive algorithm for the deterministic case that combines learners for different hypothesis classes. Let $\mathcal{H}_1, \ldots, \mathcal{H}_k \in \mathcal{Y}^{\mathcal{X}}$ be a set of KWIK-learnable hypothesis classes with KWIK bounds of $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$. That is, all the hypothesis classes share the same input and output sets. The union algorithm (Algorithm 21) can be used to KWIK-learn the joint

hypothesis class

$$\mathcal{H} \stackrel{\text{def}}{=} \bigcup_{i=1}^{k} \mathcal{H}_i.$$

Algorithm 21 Union for deterministic problems.

0: **Inputs:** $\mathcal{X}, \mathcal{Y}, \mathcal{H}_1, \ldots, \mathcal{H}_k, \mathbf{A}_1, \ldots, \mathbf{A}_k, \epsilon, \delta$.
1: Each sub-algorithm $\mathbf{A}_i$ is run with parameters $\epsilon/2$ and δ/k.
2: $\hat{A} \leftarrow \{1, 2, \cdots, k\}$.
3: **for** $t = 1, 2, \ldots$ **do**
4: Observe $x_t \in \mathcal{X}$
5: Run each $\mathbf{A}_i$ in $\hat{A}$ to obtain their predictions, $\hat{y}_{ti}$.
6: $L \leftarrow \{\hat{y}_{ti} \mid i \in \hat{A}\}$.
7: **if** $\perp \in L$ **then**
8: Predict $\hat{y}_t = \perp$ and observe $y_t \in \mathcal{Y}$
9: Send y_t to all sub-algorithms $\mathbf{A}_i$ with $\hat{y}_{ti} = \perp$
10: **else if** there exists some $y_0 \in \mathcal{Y}$ such that $\max_{y \in L} |y_0 - y| < \epsilon/2$ **then**
11: Predict $\hat{y}_t = y_0$
12: **else**
13: Predict $\hat{y}_t = \perp$ and observe $y_t \in \mathcal{Y}$
14: $\hat{A} \leftarrow \hat{A} \setminus \{i \mid |\hat{y}_{ti} - y_t| \geq \epsilon/2\}$
15: **end if**
16: **end for**

One can think of Union as a higher-level version of enumeration. It maintains a set of active sub-algorithms $\hat{A}$, one for each hypothesis class; $\hat{A}$ is initialized to $\{1, \ldots, k\}$. Given an input x_t at timestep t, the union algorithm queries each sub-algorithm $i \in \hat{A}$ to obtain a prediction $\hat{y}_{ti}$. Let $\hat{L} = \{\hat{y}_{ti} \mid i \in \hat{A}\}$.

If $\perp \in \hat{L}$, the union algorithm reports $\perp$ and obtains the correct output y_t. Any algorithm i for which $\hat{y}_{ti} = \perp$ is then sent the correct output y_t to allow it to learn. Below, we assume $\perp \notin \hat{L}$.

If all predictions in $\hat{L}$ are sufficiently consistent, as defined in Line 10 of Algorithm 21, then the prediction made in Line 11 must be ϵ-accurate because

$$|\hat{y}_t - y_t| \leq |\hat{y}_t - \hat{y}_{ti^*}| + |\hat{y}_{ti^*} - y_t| \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon,$$

where i^* denotes which hypothesis class the (unknown) target hypothesis lies in: $h^* \in \mathcal{H}_{i^*}$. With high probability, $i^* \in \hat{A}$ during the whole run of union, since predictions by $\mathbf{A}_{i^*}$ must be $\frac{\epsilon}{2}$-accurate.

If the consistency condition in Line 10 is not satisfied, the algorithm is not able to make an ϵ-accurate prediction and so returns $\perp$. Using the correct output y_t, any

algorithm that incurred a prediction error greater than or equal to $\epsilon/2$ is "killed"; that is, it is removed from the active set, as done in Line 14. Clearly, at least one sub-algorithm becomes inactive.

On each input x_t for which union reports $\perp$, either one of the sub-algorithms reported $\perp$ (at most $\sum_i B_i(\epsilon/2, \delta/k)$ times), or these sub-algorithms disagreed and at least one was removed from $\hat{A}$ (at most $k-1$ times). Finally, the total failure probability is δ, by a union bound. We have thus proved the main result of this section.

Theorem 17 *Union can KWIK-learn the joint hypothesis class* $\mathcal{H} \stackrel{\text{def}}{=} \bigcup_{i=1}^{k} \mathcal{H}_i$ *with a KWIK bound of*

$$B_{\textsf{union}}(\epsilon, \delta) = (k-1) + \sum_{i=1}^{k} B_i\left(\frac{\epsilon}{2}, \frac{\delta}{k}\right).$$

Example 14 *Let* $\mathcal{X} = \mathcal{Y} = \mathbb{R}$. *Now, define* $\mathcal{H}_1 = \{f \mid f(x) = |x - c|, c \in \mathbb{R}\}$. *That is, each function in* $\mathcal{H}_1$ *maps* x *to its distance from some unknown point* c. *We can learn* $\mathcal{H}_1$ *with a KWIK bound of 2 using a 1-dimensional version of* distance-learning. *Next, define* $\mathcal{H}_2 = \{f \mid f(x) = mx + b, m \in \mathbb{R}, b \in \mathbb{R}\}$. *That is,* $\mathcal{H}_2$ *is the set of lines. We can learn* $\mathcal{H}_2$ *with a KWIK bound of 2 using* deterministic linear-regression. *Finally, define* $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_2$, *the union of these two classes. We can use Algorithm 21 to KWIK-learn* $\mathcal{H}$ *with a KWIK bound no greater than* $2 + 2 + (2 - 1) = 5$.

Assume the first input is $x_1 = 2$. *The* union *algorithm queries the learners for* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *with* x_1 *as input and neither has any idea, so it returns* $\perp$ *and receives the feedback* $y_1 = 2$, *which it passes to the sub-algorithms. The next input is* $x_2 = 8$. *The learners for* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *still lack enough information, so it returns* $\perp$ *and sees* $y_2 = 4$, *which it passes to the sub-algorithms. Next,* $x_3 = 1$. *Now, the learner for* $\mathcal{H}_1$ *unambiguously computes* $c = 4$, *because that's the only interpretation consistent with the first two examples (*$|2 - 4| = 2$, $|8 - 4| = 4$*), so it returns* $|1 - 4| = 3$. *On the other hand, the learner for* $\mathcal{H}_2$ *unambiguously computes* $m = 1/3$ *and* $b = 4/3$, *because that's the only interpretation consistent with the first two examples (*$2 \times 1/3 + 4/3 = 2$, $8 \times 1/3 + 4/3 = 4$*), so it returns* $1 \times 1/3 + 4/3 = 5/3$. *Since the two sub-algorithms disagree, the* union *algorithm returns* $\perp$ *one last time and finds out that* $y_3 = 3$. *It makes all future predictions (accurately) using the algorithm for* $\mathcal{H}_1$. *The total number*

of $\perp$s returned is 3.

6.5 Noisy Union

In this section, we provide a powerful and general algorithm that extends union to work with noisy observations as well. In a stochastic problem, we cannot eliminate a candidate hypothesis simply based on a single mistake it makes (as we did in union). Instead, we use a scoring function to measure the *cumulative error* of each hypothesis, which gets eliminated from the version space when its cumulative error becomes too large. A similar technique is used for hypothesis testing by Kearns and Schapire [1994], who consider a PAC-style learning model for probabilistic concepts.

Let $\mathcal{X}$ be the input set, $\mathcal{Y} = [0, 1]$ be the output set, $\mathcal{Z} = \mathbb{B}$ a binary observation set, and $\mathcal{H}_1, \ldots, \mathcal{H}_k$ be a set of KWIK-learnable hypothesis classes with KWIK bounds of $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$ where $\mathcal{H}_i \subseteq \mathcal{Y}^{\mathcal{X}}$ for all i. That is, all the hypothesis classes share the same input and output sets. The noisy union algorithm (Algorithm 22) can be used to KWIK-learn the joint hypothesis class $\mathcal{H} \stackrel{\text{def}}{=} \bigcup_i \mathcal{H}_i$ with a KWIK bound of

$$B_{\text{noisy union}}(\epsilon, \delta) = \mathcal{O}\left(\frac{k}{\epsilon^2} \ln \frac{k}{\delta}\right) + \sum_{i=1}^{k} B_i\left(\frac{\epsilon}{8}, \frac{\delta}{k+1}\right).$$

Furthermore, we provide a lemma at the end of the section showing that noisy union is optimal up to a constant factor.

We first sketch the special case of $k = 2$ to explain the intuition of noisy union, and then extend the analysis to the general case. This algorithm is similar to the union algorithm, except that it has to deal with noisy observations. The algorithm proceeds by running the KWIK sub-algorithms, denoted $\mathbf{A}_1$ and $\mathbf{A}_2$, using parameters (ϵ_0, δ_0), as sub-algorithms for each of the $\mathcal{H}_i$ hypothesis classes, where $\epsilon_0 = \epsilon/4$ and $\delta_0 = \delta/3$. Given an input x_t in timestep t, it queries each sub-algorithm $\mathbf{A}_i$ to obtain a prediction $\hat{y}_{ti}$. Let $\hat{L}_t = \{\hat{y}_{t1}, \hat{y}_{t2}\}$ be the set of predictions.

If $\perp \in \hat{L}_t$, noisy union reports $\perp$, obtains an observation $z_t \in \mathcal{Z}$, and sends it to sub-algorithms i if $\hat{y}_{ti} = \perp$ to allow it to learn. In the following, we focus on the remaining cases where $\perp \notin \hat{L}_t$.

If $|\hat{y}_{t1} - \hat{y}_{t2}| \leq 4\epsilon_0$, then these two predictions are sufficiently consistent, so with high probability the prediction $\hat{p}_t = (\hat{y}_{t1} + \hat{y}_{t2})/2$ is ϵ-close to $y_t = \Pr(z_t = 1)$. The reason is that, since one of the predictions, say $\hat{y}_{t1}$, deviates from y_t by at most ϵ_0 with probability at least $1 - \delta/3$, and so the triangle inequality implies

$$|\hat{p}_t - y_t| \leq |\hat{p}_t - \hat{y}_{t1}| + |\hat{y}_{t1} - \hat{y}_t| = \frac{|\hat{y}_{t1} - \hat{y}_{t2}|}{2} + |\hat{y}_{t1} - \hat{y}_t| \leq 2\epsilon_0 + \epsilon_0 < \epsilon.$$

If $|\hat{y}_{t1} - \hat{y}_{t2}| > 4\epsilon_0$, then the individual predictions are not sufficiently consistent for noisy union to make an ϵ-accurate prediction. Thus, it reports $\perp$ and needs to know which sub-algorithm provided an inaccurate response. But, since the observations are noisy in this problem, it cannot eliminate h_i on the basis of a single observation. Instead, it maintains the total squared prediction error for every sub-algorithm i: $e_i = \sum_{t \in \mathcal{D}} (\hat{y}_{ti} - z_t)^2$, where $\mathcal{D} = \{t \mid |\hat{y}_{t1} - \hat{y}_{t2}| > 4\epsilon_0\}$ is the set of trials in which the sub-algorithms gave inconsistent predictions. It can be shown using simple algebra that $\mathbf{E}[e_1] < \mathbf{E}[e_2]$ if $h^* \in \mathcal{H}_1$, and vice versa. Our last step is to show e_i provides a robust measure for eliminating invalid predictors when $|\mathcal{D}|$ is sufficiently large.

Using Hoeffding's inequality and some algebra, we find

$$\Pr(\ell_1 > \ell_2) \leq \exp\left(-\frac{\sum_{t \in \mathcal{D}} |\hat{y}_{t1} - \hat{y}_{t2}|^2}{8}\right) \leq \exp\left(-2\epsilon_0^2 |\mathcal{D}|\right).$$

Setting the righthand side to be $\delta/3$ and solving for $|\mathcal{D}|$, we have

$$|\mathcal{D}| = \frac{1}{2\epsilon_0^2} \ln \frac{3}{\delta} = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right).$$

Since each $\mathbf{A}_i$ succeeds with probability $1 - \delta/3$, and the comparison of e_1 and e_2 also succeeds with probability $1 - \delta/3$, a union bound implies that the noisy union algorithm succeeds with probability at least $1 - \delta$. All $\perp$s are either from a sub-algorithm (at most $\sum_i B_i(\epsilon_0, \delta_0)$) or from the noisy union algorithm itself ($\mathcal{O}(1/\epsilon^2 \ln(1/\delta))$).

The general case where $k > 2$ can be reduced to the $k = 2$ case by pairing the k learners and running noisy union described above on each pair. Here, each sub-algorithm is run with parameters $\epsilon/8$ and $\delta/(k+1)$. The algorithm is formally described in Algorithm 22. Although there are $\binom{k}{2} = \mathcal{O}(k^2)$ pairs, a similar but more careful analysis can reduce the dependence of the KWIK bound on k from quadratic to linearithmic, leading to Theorem 18.

Algorithm 22 Noisy union

Inputs: $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}_1, \ldots, \mathcal{H}_k, \mathbf{A}_1, \ldots, \mathbf{A}_k, \epsilon, \delta, m$.
Run each sub-algorithm $\mathbf{A}_i$ with parameters $\epsilon/8$ and $\delta/(k+1)$.
$R \leftarrow \{1, 2, \cdots, k\}$ {version space of hypothesis classes}
for all $1 \leq i < j \leq k$ **do**
  $c_{ij} \leftarrow 0$
  $\Delta_{ij} \leftarrow 0$
end for
for timestep $t = 1, 2, \ldots$ **do**
  Observe $x_t \in \mathcal{X}$
  Run each $\mathbf{A}_i$ to obtain its prediction, $\hat{y}_{ti}$
  if $\hat{y}_{ti} = \perp$ for some $i \in R$ **then**
    Predict $\hat{y}_t = \perp$ and observe $z_t \in \mathcal{Z}$
    Send z_t to all sub-algorithms $\mathbf{A}_i$ with $\hat{y}_{ti} = \perp$
  else
    if $|\hat{y}_{ti} - \hat{y}_{tj}| < \epsilon$ for all $i, j \in R$ **then**
      Predict the midpoint of the predictions: $\hat{y}_t = (\max_{i \in R} \hat{y}_{ti} + \min_{i \in R} \hat{y}_{ti})/2$
    else
      Predict $\hat{y}_t = \perp$ and observe $z_t \in \mathcal{Z}$
      for all $i, j \in R$ such that $i < j$ and $|\hat{y}_{ti} - \hat{y}_{tj}| \geq \epsilon/2$ **do**
        $c_{ij} \leftarrow c_{ij} + 1$
        $\Delta_{ij} \leftarrow \Delta_{ij} + (\hat{y}_{ti} - z_t)^2 - (\hat{y}_{tj} - z_t)^2$
        if $c_{ij} = m$ **then**
          $R \leftarrow R \setminus \{I\}$ where $I = i$ if $\Delta_{ij} > 0$ and $I = j$ otherwise.
        end if
      end for
    end if
  end if
end for

Theorem 18 *Noisy union can KWIK-learn the combined hypothesis class* $\mathcal{H} \stackrel{\text{def}}{=} \mathcal{H}_1 \cup \cdots \cup \mathcal{H}_k$ *with a KWIK bound of*

$$B_{\text{noisy union}}(\epsilon, \delta) = \mathcal{O}\left(\frac{k}{\epsilon^2} \ln \frac{k}{\delta}\right) + \sum_{i=1}^{m} B_i\left(\frac{\epsilon}{8}, \frac{\delta}{k+1}\right)$$

when the parameter m *is set appropriately:*

$$m = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{k}{\delta}\right).$$

Example 15 *[Li et al. 2009b] There are* k *meteorologists in a town. Every morning, each of them makes a prediction about the chances of rain of that day. At the end of the day, we observe and record whether it rained or not. Such historical data may be used by* noisy union *to identify an accurate meteorologist (assuming he exists) after polynomially many days.*

Finally, we provide a lower bound showing the optimality of noisy union for combining hypothesis classes. To simplify exposition, we separate the KWIK bounds of sub-algorithms from the KWIK bound used to select the right sub-algorithm in noisy union. To do so, we assume $B_i(\epsilon, \delta)$ are all zero in order to show the optimality of noisy union in terms of *combining* hypothesis classes. The proof is left in §6.7.2.

Theorem 19 *Suppose all sub-algorithms have been KWIK-learned beforehand; that is, $B_i(\epsilon, \delta) = 0$. Then, a KWIK lower bound for the noisy union is*

$$\Omega\left(\frac{k}{\epsilon^2}\ln\frac{k}{\delta}\right).$$

6.6 Case Study: Multivariate Normal Learning

In this section, we give a non-trivial study of KWIK-learning multivariate normal distributions to illustrate some of the combination techniques presented in previous sections. This section extends and generalizes a number of previous results in the literature [Abbeel and Ng 2005; Strehl and Littman 2008b; Brunskill et al. 2008].

Recall that a multivariate normal distribution, $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, has the following probability density function: for any $\mathbf{x} \in \mathbb{R}^n$,

$$\varphi(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{(2\pi)^{n/2}\sqrt{\det\Sigma}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

where $\boldsymbol{\mu} = [\mu_i]_i \in \mathbb{R}^n$ is the mean vector, and $\Sigma = [\sigma_{ij}]_{ij} \in \mathbb{R}^{n\times n}$ is the covariance matrix. We assume Σ is non-singular. The problem of KWIK-learning a multivariate normal distribution can be formulated by defining the following: the input set $\mathcal{X}$ is a singleton indicating there is a single normal distribution to learn,

$$\begin{aligned}\mathcal{Y} &= \mathcal{H} = \{(\boldsymbol{\mu}, \Sigma)\} \subseteq \mathbb{R}^n \times \mathbb{R}^{n\times n},\\ \mathcal{Z} &= \mathbb{R}^n,\end{aligned}$$

and the discrepancy function is defined as the total variation, as usual:

$$|\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1), \mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)| = \mathrm{d}_{\mathrm{var}}\left(\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1), \mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)\right).$$

The normal-learning algorithm (Algorithm 23) is very intuitive. Before making predictions, it simply outputs $\perp$ to collect a sufficient amount of samples to obtain highly

accurate maximum-likelihood predictions of the mean vector and the covariance matrix. It is not difficult to show that, after m observations, the estimate of the normal distribution computed by Algorithm 23 coincides with the maximum-likelihood estimate:

$$\begin{aligned} \hat{\boldsymbol{\mu}} &= \frac{1}{m}\sum_{t=1}^{m}\mathbf{z}_t \\ \hat{\Sigma} &= \frac{1}{m}\sum_{t=1}^{m}(\mathbf{z}_t - \hat{\boldsymbol{\mu}})(\mathbf{z}_t - \hat{\boldsymbol{\mu}})^\top. \end{aligned}$$

The algorithm has $\mathcal{O}(n^2)$ per-step computational complexity and space complexity.

Algorithm 23 Normal-learning.

Inputs: $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}, \epsilon, \delta, m$.
$\hat{\boldsymbol{\mu}} \leftarrow \mathbf{0}_n$, $\hat{\Sigma} \leftarrow O_{n\times n}$.
for $t = 1, 2, \ldots$ **do**
  Observe $x_t \in \mathcal{X}$.
  if $t \leq m$ **then**
    Predict $\hat{y}_t = \perp$ to observe $\mathbf{z}_t \in \mathcal{Z}$.
    $\hat{\boldsymbol{\mu}} \leftarrow \hat{\boldsymbol{\mu}} + \mathbf{z}_t$.
    $\hat{\Sigma} \leftarrow \hat{\Sigma} + \mathbf{z}_t\mathbf{z}_t^\top$.
    if $t = m$ **then**
      $\hat{\boldsymbol{\mu}} \leftarrow \hat{\boldsymbol{\mu}}/m$.
      $\hat{\Sigma} \leftarrow \hat{\Sigma}/m - \hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^\top$.
    end if
  else
    Predict $\hat{y}_t = (\hat{\boldsymbol{\mu}}, \hat{\Sigma})$.
  end if
end for

To simplify notation, in the rest of this section as well as §6.7.3 and §6.7.4, we suppress the timestep t in our notation, and use z_i to denote the i-th component in vector $\mathbf{z}$. The (i, j)-entries of Σ and $\hat{\Sigma}$ are denoted σ_{ij} and $\hat{\sigma}_{ij}$, respectively.

In §5.3.3, we discuss how to KWIK-learn $\boldsymbol{\mu}$ when $n = 1$ and Σ is known. In the general learning problem defined above, a few challenges arise:

- How can the learner robustly learn $\boldsymbol{\mu}$ to a desired precision?
- How can the learner robustly learn Σ to a desired precision?
- How can the learner translate prediction errors in $\boldsymbol{\mu}$ and Σ to the total variation defined above?

The first two questions are relatively easy to solve given the tools we have developed so far, while the last is rather difficult, requiring simplifying assumptions in previous

work: Abbeel and Ng [2005] and Strehl and Littman [2008b] assume that $\Sigma = \sigma^2 I_n$ for some known σ^2 and identity matrix I_n; Brunskill et al. [2008] assume Σ is diagonal. Here, we replace all these assumptions by a mild one that is satisfied in most reasonable applications. In particular, we assume that there exists a constant, $B \geq 1$, such that

$$\max_{i,j} |\sigma_{ij}| \quad \leq \quad B.$$

Although our analysis below is complicated, it reduces to essentially the same bounds (modulo constant factors) of Abbeel and Ng [2005] and Brunskill et al. [2008] under respective assumptions.

The three challenges above are addressed by three lemmas (17–19), respectively. We should point out that there are opportunities to prove tightened bounds than those given below, but the main point here is to illustrate how we may guarantee polynomial KWIK bounds by combining individual KWIK learners in a non-trivial example, and how one may link discrepancy in individual output spaces to the joint output spaces (*c.f.*, Lemma 19). A refined analysis is left for future investigation.

Lemma 17 *It suffices to set*

$$m = \frac{2n^2 B}{\epsilon^2 \delta} = \mathcal{O}\left(\frac{n^2 B}{\epsilon^2 \delta}\right).$$

to guarantee $\|\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}\|_2 \leq \epsilon$ *with probability at least* $1 - \delta/2$.

Proof. A straightforward application of output combination to Equation 5.5 using $\alpha = 1/\sqrt{n}$. Note that we have made use of the assumption that $\mathbf{Var}[z_i] = \sigma_{ii} \leq B$. □

Lemma 18 *Assume* $\max_i |\hat{\mu}_i - \mu_i| \leq \epsilon$ *for* $\epsilon < 1/3$. *Then, it suffices to set*

$$m = \frac{8n^2 B^2}{\epsilon^2 \delta} = \mathcal{O}\left(\frac{n^2 B^2}{\epsilon^2 \delta}\right).$$

to guarantee $|\hat{\sigma}_{ij} - \sigma_{ij}| \leq \epsilon$ *for all* i, j *with probability at least* $1 - \delta/2$.

Proof. The proof is again a straightforward application of output combination to Lemma 54 using $\alpha = 1$. Specifically, we may KWIK-learn each element in the matrix Σ to within ϵ-accuracy, and then use a union bound to KWIK-learn all elements in Σ. The

only difficult step is to get an upper bound of the variance of the maximum-likelihood estimate of σ_{ij}, which is solved by Lemma 27 in §6.7.3. □

The last lemma relates the discrepancy metrics for $\boldsymbol{\mu}$ and Σ to the total variation of normal distributions. Its proof is rather complicated, to which §6.7.4 is devoted.

Lemma 19 *Assume* $\|\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}\|_2 \leq \epsilon_1$, $\max_{i,j} |\hat{\sigma}_{ij} - \sigma_{ij}| \leq \epsilon_2$, *and* $n\epsilon_2 \left\|\Sigma^{-1}\right\|_1 < 1$. *Then,*

$$\mathrm{d_{var}}\left(\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\Sigma}), \mathcal{N}(\boldsymbol{\mu}, \Sigma)\right) \leq \frac{\epsilon_1}{\sqrt{\lambda_n}} + \sqrt{\frac{n^2\epsilon_2}{\lambda_n} + \frac{2n^3 B\epsilon_2}{\lambda_n^2 - n^{3/2}\lambda_n\epsilon_2}},$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n > 0$ *are the* n *eigenvalues of the covariance matrix* Σ. *Note that* $\epsilon_2 \leq \lambda_n n^{-3/2}$ *suffices to guarantee* $n\epsilon_2 \left\|\Sigma^{-1}\right\|_1 < 1$ *due to Lemma 28.*

Putting all these three pieces together, we may prove the following KWIK bound for Algorithm 23:

Theorem 20 *Algorithm 23 has the following KWIK bound:*

$$B_{\textsf{normal-learning}}(\epsilon, \delta) = \mathcal{O}\left(\frac{n^8 B^4}{\lambda_n^4 \epsilon^4 \delta}\right),$$

where $\lambda_n > 0$ *is the smallest eigenvalue of the covariance matrix* Σ.

Proof. To guarantee $\mathrm{d_{var}}\left(\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\Sigma}), \mathcal{N}(\boldsymbol{\mu}, \Sigma)\right) \leq \epsilon$, it follows from Lemma 19 that it suffices to have

$$\begin{aligned}
\frac{\epsilon}{2} &\geq \frac{\epsilon_1}{\sqrt{\lambda_n}} \\
\frac{\epsilon^2}{8} &\geq \frac{n^2\epsilon_2}{\lambda_n} \\
\frac{\epsilon^2}{8} &\geq \frac{n^3 B\epsilon_2}{\lambda_n^2 - n^{3/2}\lambda_n\epsilon_2}.
\end{aligned}$$

Solving the three inequalities for ϵ_1 and ϵ_2, we obtain

$$\epsilon_1 \leq \frac{\sqrt{\lambda_n}\epsilon}{2} \tag{6.1}$$

$$\epsilon_2 \leq \frac{\lambda_n\epsilon^2}{8n^2} \tag{6.2}$$

$$\epsilon_2 \leq \frac{\lambda_n^2\epsilon^2}{8n^3 B + n^{3/2}\lambda_n\epsilon^2}. \tag{6.3}$$

Equations 6.2 and 6.3 imply that

$$\epsilon_2 \leq \min\left\{\frac{\lambda_n\epsilon^2}{8n^2}, \frac{\lambda_n^2\epsilon^2}{8n^3B + n^{3/2}\lambda_n\epsilon^2}\right\} = \frac{\lambda_n^2\epsilon^2}{8n^3B + n^{3/2}\lambda_n\epsilon^2}, \tag{6.4}$$

where equality holds because $\lambda_n \leq B$ (*c.f.*, Lemma 28). We now apply Lemma 17 and Lemma 18 to Equation 6.1 and Equation 6.4 to obtain

$$m = \max\left\{\frac{16n^2B}{\lambda_n\epsilon^2\delta}, \frac{8n^2B^2(8n^3B + n^{3/2}\lambda_n\epsilon^2)^2}{\lambda_n^4\epsilon^4\delta}\right\} = \mathcal{O}\left(\frac{n^8B^4}{\lambda_n^4\epsilon^4\delta}\right).$$

Finally, a straightforward application of the union bound implies that the probability of failing to guarantee ϵ-KL-divergence is at most $\delta/2+\delta/2 = \delta$, which proves the desired KWIK bound. □

6.7 Proofs

This section provides detailed proofs of all technical lemmas used in this chapter.

6.7.1 Proof of Theorem 18

Without loss of generality, assume $h^* \in \mathcal{H}_1$, and so $|\hat{y}_{t1} - y_t| \leq \epsilon/8$ whenever $\hat{y}_{t1} \neq \perp$. The following lemma says the accuracy requirement of Definition 11 is satisfied.

Lemma 20 *If $\hat{y}_t \neq \perp$, then $|\hat{y}_t - y_t| < \epsilon$.*

Proof. By assumption, $|\hat{y}_{t1} - y_t| \leq \epsilon/8$. Since the midpoint prediction $\hat{y}_t$ differs from $\hat{y}_{t1}$ by at most $\epsilon/2$, the prediction error can be bounded using the triangle inequality:

$$|\hat{y}_t - y_t| \leq |\hat{y}_t - \hat{y}_{t1}| + |\hat{y}_{t1} - y_t| \leq \frac{\epsilon}{2} + \frac{\epsilon}{8} < \epsilon.$$

□

We next show that the sample-complexity requirement of Definition 11 is also satisfied. Note that the total number of $\perp$s returned by noisy union is the number of timesteps Lines 11 and 17 of Algorithm 22 are executed. Since Line 11 can be executed at most $\sum_i B(\epsilon/8, \delta/(k+1))$ times, all that remains is to show that Line 17 cannot be executed many times. To do this, we first prove the following three lemmas.

Lemma 21 *Whenever Line 17 is executed, c_{1i} and Δ_{1i} will be updated for at least one i in $\{2, 3, \cdots, k\}$. Consequently, Line 17 is executed on at most $m(k-1)$ timesteps.*

PROOF. Suppose at timestep t noisy union predicts $\hat{y}_t = \perp$ because $|\hat{y}_{ti} - \hat{y}_{tj}| \geq \epsilon$ for some $1 \leq i < j \leq k$. Then, by the triangle inequality $|a| + |b| \geq |a - b|$, we have that $|\hat{y}_{t1} - \hat{y}_{ti}| + |\hat{y}_{t1} - \hat{y}_{tj}| \geq |\hat{y}_{ti} - \hat{y}_{tj}| \geq \epsilon$, which implies at least one of $|\hat{y}_{t1} - \hat{y}_{ti}| \geq \epsilon/2$ or $|\hat{y}_{t1} - \hat{y}_{tj}| \geq \epsilon/2$ is true. Hence, either c_{1i} and Δ_{1i}, or c_{1j} and Δ_{1j}, or both, are updated. □

We next turn to decide an appropriate value of m to guarantee that the correct hypothesis is not ruled out with high probability.

Lemma 22 *Let i be the index given in the Lemma 21 such that $|\hat{y}_{t1} - \hat{y}_{ti}| \geq \epsilon/2$. On average, Δ_{1i} is decremented by at least $\epsilon^2/8$.*

PROOF. By definition and simple algebra, the expected *increment* of Δ_{1i} is:

$$
\begin{aligned}
&\mathbf{E}_{z_t \sim y_t}\left[(\hat{y}_{t1} - z_t)^2 - (\hat{y}_{ti} - z_t)^2\right] \\
&= \mathbf{E}_{z_t \sim y_t}\left[(\hat{y}_{t1} - \hat{y}_{ti})(\hat{y}_{t1} + \hat{y}_{ti} - 2z_t)\right] \\
&= y_t(\hat{y}_{t1} - \hat{y}_{ti})(\hat{y}_{t1} + \hat{y}_{ti} - 2) + (1 - y_t)(\hat{y}_{t1} - \hat{y}_{ti})(\hat{y}_{t1} + \hat{y}_{ti}) \\
&= -(\hat{y}_{t1} - \hat{y}_{ti})^2 + 2(\hat{y}_{t1} - \hat{y}_{ti})(\hat{y}_{t1} - y_t) \\
&\leq -(\hat{y}_{t1} - \hat{y}_{ti})^2 + 2\left|(\hat{y}_{t1} - \hat{y}_{ti})(\hat{y}_{t1} - y_t)\right| \\
&= -(\hat{y}_{t1} - \hat{y}_{ti})^2 + 2\left|\hat{y}_{t1} - \hat{y}_{ti}\right|\left|\hat{y}_{t1} - y_t\right| \\
&\leq -(\hat{y}_{t1} - \hat{y}_{ti})^2 + \frac{\epsilon}{4}\left|\hat{y}_{t1} - \hat{y}_{ti}\right| \\
&= \left|\hat{y}_{t1} - \hat{y}_{ti}\right|\left(-\left|\hat{y}_{t1} - \hat{y}_{ti}\right| + \frac{\epsilon}{4}\right) \\
&\leq \frac{\epsilon}{2}\left(-\frac{\epsilon}{2} + \frac{\epsilon}{4}\right) = -\frac{\epsilon^2}{8}.
\end{aligned}
$$

□

Lemma 23 *Each change of Δ_{1i} in Line 20 is at most $2\left|\hat{y}_{t1} - \hat{y}_{ti}\right|$.*

PROOF. Using simple algebra, we have

$$
\left|(\hat{y}_{t1} - z_t)^2 - (\hat{y}_{ti} - z_t)^2\right| = \left|(\hat{y}_{t1} - \hat{y}_{ti})(\hat{y}_{t1} + \hat{y}_{ti} - 2z_t)\right| = \left|\hat{y}_{t1} - \hat{y}_{ti}\right|\left|\hat{y}_{t1} + \hat{y}_{ti} - 2z_t\right|.
$$

It can be verified that $|\hat{y}_{t1} + \hat{y}_{ti} - 2z_t| < 2$ for $z_t \in \{0, 1\}$, and the lemma follows. □

Based on Lemma 22 and 23, we can decide the value of m. To simplify notation, assume without loss of generality that $|\hat{y}_{t1} - \hat{y}_{ti}| \geq \epsilon/2$ for the first m timesteps; namely, Δ_{1i} changes in every timestep until one of $\mathbf{A}_1$ and $\mathbf{A}_i$ is eliminated in Line 22 at the end of timestep m. Applying Hoeffding's inequality to the martingale Δ_{1i}, we have at the end of timestep m that

$$\begin{aligned}
\Pr\left(\Delta_{1i} \geq 0\right) &\leq \Pr\left(\Delta_{1i} - \mathbf{E}[\Delta_{1i}] \geq \sum_{t=1}^{m} \frac{\epsilon_t^2}{2}\right) \\
&\leq \exp\left(-\frac{\left(\sum_{t=1}^{m} \epsilon_t^2/2\right)^2}{2\sum_{t=1}^{\top}(2\epsilon_t)^2}\right) \\
&= \exp\left(-\frac{\sum_{t=1}^{m} \epsilon_t^2}{32}\right) \\
&\leq \exp\left(-\frac{m\epsilon^2}{128}\right)
\end{aligned}$$

where the first inequality is due to Lemma 22, the second due to Hoeffding's inequality and Lemma 23, and the last due to the fact that $\epsilon_t \geq \epsilon/2$. Setting the last expression to δ/k^2, we can solve for m:

$$m = \frac{128}{\epsilon^2} \ln \frac{k^2}{\delta}.$$

By the union bound, we have that

$$\Pr\left(\Delta_{1i} > 0 \text{ for any } i \in \{2, 3, \ldots, k\}\right) \leq (k-1)\frac{\delta}{k^2} < \frac{\delta}{k+1}.$$

That is, the probability that the noisy union algorithm ends up with an incorrect hypothesis is tiny if every sub-algorithm succeeds. Applying the union bound again with the k sub-algorithms, $\mathbf{A}_i$, each failing with probability at most $\delta/(k+1)$, we conclude that the total failure probability of noisy union is at most δ.

Using the m value derived above and Lemma 21, the KWIK bound of noisy union is

$$m(k-1) + \sum_{i=1}^{m} B_i\left(\frac{\epsilon}{8}, \frac{\delta}{k+1}\right) = \frac{128(k-1)}{\epsilon^2} \ln \frac{k^2}{\delta} + \sum_{i=1}^{m} B_i\left(\frac{\epsilon}{8}, \frac{\delta}{k+1}\right)$$

6.7.2 Proof of Theorem 19

The proof is through a reduction to the multi-armed bandit problem (*c.f.*, §4.3.1). In particular, we will construct a problem such that a KWIK learner for $\mathcal{H} \stackrel{\text{def}}{=} \bigcup_{i=1}^{k} \mathcal{H}_i$

has to solve $k-1$ instances of two-armed bandit problems, each of which requires a sample complexity lower bound in Lemma 11.

Let $\mathbf{A}$ be a hypothetical KWIK algorithm that solves the k-meteorologist problem with a sample complexity $B(\epsilon, \delta)$. Given any ϵ and δ, we define the following KWIK problem: $\mathcal{X} = \{1, 2, \ldots, k-1\}, \mathcal{Y} = [0, 1], \mathcal{Z} = \mathbb{B}$, and $\mathcal{H}_i = \{h_i\}$ for $i = 1, 2, \ldots, k$. The target function $h^* = h_k$. The set of k meteorologists are defined by

$$h_i(j) \stackrel{\text{def}}{=} 1 - \epsilon + 2\epsilon \cdot \mathbb{I}(i \leq j).$$

In other words, the set of meteorologists are constructed such that h_i makes accurate predictions for inputs $x \in \{i+1, i+2, \ldots, k-1\}$, and non-$\epsilon$-accurate predictions for inputs $x \in \{1, 2, \ldots, i\}$. We now construct a sequence of inputs in the following way: fix a constant $\tau \in \mathbb{N}$, and let

$$x_1 = x_2 = \cdots = x_\tau = 1$$

$$x_{\tau+1} = x_{\tau+2} = \cdots = x_{2\tau} = 2$$

$$\cdots$$

$$x_{(k-2)\tau+1} = x_{(k-2)\tau+2} = \cdots = k-1.$$

That is, the inputs consists of $k-1$ blocks of the same inputs. As h^* is the target function, we always have $z_t = 1$ with probability $1+\epsilon$ and 0 with probability $1-\epsilon$.

Without loss of generality, we may assume that, for each input block, $\mathbf{A}$ returns $\perp$ for a while and then switches to non-$\perp$ predictions. As long as $\tau > \zeta(\epsilon, \delta)$, $\mathbf{A}$ should make non-$\perp$ predictions at the end of each input block.

In the i-th input block where inputs are always i, the algorithm should, with high probability, eventually discover h_i is not ϵ-correct. Note that the two-armed bandit problem used by Mannor and Tsitsiklis [2004] can be reduced to the problem of finding out whether $\mathbf{E}[z_t]$ equals $1-\epsilon$ or $1+\epsilon$; in this reduction, the sample complexity for the two-armed bandit becomes the number of $\perp$s returned by $\mathbf{A}$. Thus, the same lower bound in Lemma 11 applies to $\mathbf{A}$, with which we can prove the following lemma.

Lemma 24 *After returning $\perp$ for m_i times on input $i \in \mathcal{X}$, the probability that* $\mathbf{A}$ fails

to discover the non-ϵ-accuracy of h_i is at least

$$p(m_i) \stackrel{\text{def}}{=} c_2 \exp\left(-c_1 m_i \epsilon^2\right),$$

for some constants $c_1, c_2 \in \mathbb{R}_+$.

PROOF. Lemma 11 implies a lower bound of the number of samples to discover the non-ϵ-optimality of h_i with probability $1 - \delta_i$:

$$m_i \geq \frac{1}{c_1 \epsilon^2} \ln \frac{c_2}{\delta_i}$$

for some constants $c_1, c_2 \in \mathbb{R}_+$. Reorganizing terms to solve for δ_i gives the desired result. □

Given the lower bound for any individual input $x \in \mathcal{X}$, the remaining question is how to get a KWIK lower bound for the whole run of $\mathbf{A}$. Recall that δ is the probability of failing on *any* input:

$$\begin{aligned}
\delta &= \Pr(\mathbf{A} \text{ fails to eliminate } h_i \text{ for } \textit{some } i < k) \\
&= 1 - \Pr(\mathbf{A} \text{ eliminates } h_i \text{ for } \textit{all } i < k) \\
&= 1 - \prod_{i=1}^{k-1} \Pr(\mathbf{A} \text{ eliminates } h_i) \\
&= 1 - \prod_{i=1}^{k-1} (1 - \Pr(\mathbf{A} \text{ fails to eliminate } h_i)) \\
&\geq 1 - \prod_{i=1}^{k-1} (1 - p(m_i)),
\end{aligned}$$

where the third equality is due to the statistical independence between input blocks, and the last inequality is due to Lemma 24.

Two technical lemmas (Lemmas 25 and 26 below) are needed to further lower bound the last expression above, which in turn lower-bounds δ and allows one to infer a lower bound of $B(\epsilon, \delta) = m_1 + m_2 + \cdots + m_{k-1}$. Specifically, we apply Lemma 25 with $c = c_2$ and $\Delta = \exp(-c_1 \epsilon^2)$ and obtain[1]

$$\delta \geq 1 - \left(1 - c_2 \exp\left(-\frac{c_1 B(\epsilon, \delta) \epsilon^2}{k-1}\right)\right)^{k-1}.$$

[1]If $c_2 \geq 1$, we can always replace it with another constant $c_2' < 1$ without violating Lemma 11.

The theorem then follows immediately from Lemma 26 using $N = k - 1$ and $\eta = \epsilon^2$.

We now state and prove the two technical lemmas.

Lemma 25 *Let c and Δ be constants in $(0, 1)$. Under the constraints $\sum_i m_i \leq \zeta$ and $m_i > 0$ for all i, the function*

$$f(m_1, m_2, \ldots, m_N) = 1 - \prod_{i=1}^{N} (1 - c\Delta^{m_i})$$

is minimized when $m_1 = m_2 = \cdots = m_N = \frac{\zeta}{N}$. Therefore,

$$f(m_1, m_2, \ldots, m_N) \geq 1 - (1 - c\Delta^{\frac{\zeta}{N}})^N.$$

PROOF. Since $f(m_1, \ldots, m_N) \in (0, 1)$, finding the *minimum* of f is equivalent to finding the *maximum* of the following function:

$$g(m_1, m_2, \ldots, m_N) = \ln(1 - f(m_1, m_2, \ldots, m_N)) = \sum_{i=1}^{N} \ln(1 - c\Delta^{m_i}),$$

under the same constraints. Due to the concavity of $\ln(\cdot)$, we have

$$g(m_1, m_2, \ldots, m_N) \leq N \ln \left(\frac{1}{N} \sum_{i=1}^{N} (1 - c\Delta^{m_i}) \right) = N \ln \left(1 - \frac{c}{N} \sum_{i=1}^{N} \Delta^{m_i} \right).$$

Finally, we use the fact that the arithmetic mean is no less than the geometric mean to further simplify the upper bound of g:

$$g(m_1, m_2, \ldots, m_N) \leq N \ln \left(1 - c\Delta^{\frac{1}{N} \sum_{i=1}^{N} m_i} \right) \leq N \ln(1 - c\Delta^{\frac{\zeta}{N}}).$$

Equality holds in both inequalities above when $m_1 = m_2 = \cdots = m_N = \frac{\zeta}{N}$. □

Lemma 26 *If*

$$\delta \geq 1 - \left(1 - c_2 \exp\left(-\frac{c_1 \zeta \eta}{N} \right) \right)^N \tag{6.5}$$

for some constants $c_1, c_2 \in \mathbb{R}_+$, then

$$\zeta = \Omega \left(\frac{N}{\eta} \ln \frac{N}{\delta} \right).$$

PROOF. Reorganizing terms in Equation 6.5 gives

$$1 - c_2 \exp\left(-\frac{c_1 \zeta \eta}{N}\right) \geq (1-\delta)^{\frac{1}{N}}.$$

The function $(1-\delta)^{1/\delta}$ is a decreasing function of δ for $0 < \delta < 1$, and $\lim_{\delta \to 0^+}(1-\delta)^{1/\delta} = 1/e$. Therefore, as long as δ is less than some constant $c_3 \in (0,1)$, we will have

$$(1-\delta)^{\frac{1}{N}} = \left((1-\delta)^{\frac{1}{\delta}}\right)^{\frac{\delta}{N}} \geq (c_4)^{\frac{\delta}{N}} = \exp\left(-\frac{c_5 \delta}{N}\right),$$

where $c_4 = (1-c_3)^{1/c_3} \in \left(0, \frac{1}{e}\right)$ and $c_5 = \ln \frac{1}{c_4} \in (1, \infty)$ are two constants. It is important to note that c_3 (and thus c_4 and c_5) does not depend on ϵ or N or A. Now, apply the inequality $e^x \geq 1 + x$ for $x = -c_5\delta/N \in \mathbb{R}$ to get $\exp(-c_5\delta/N) \geq 1 - c_5\delta/N$. The above chain of inequalities results in:

$$1 - c_2 \exp\left(-\frac{c_1 \zeta \eta}{N}\right) \geq 1 - \frac{c_5 \delta}{N}.$$

Solving this inequality for ζ yields the desired lower bound:

$$\zeta \geq \frac{N}{c_1 \eta} \ln \frac{c_2 N}{c_5 \delta}.$$

□

6.7.3 Proof of Lemma 27

Lemma 27 *Let $\hat{\mu}_i$ and $\hat{\mu}_j$ be ϵ-accurate estimates of μ_i and μ_j in a multivariate normal distribution, $\mathcal{N}(\mu, \Sigma)$, from which $\mathbf{z}$ is drawn. Define $s_{ij} \stackrel{\text{def}}{=} (z_i - \hat{\mu}_i)(z_j - \hat{\mu}_j)$ and assume $\epsilon < 1/3$. Then,*

$$\mathbf{Var}[s_{ij}] \leq 4B^2.$$

PROOF. We first upper bound $\mathbf{Var}[s_{ij}]$ by a well-known fact in probability theory:

$$\mathbf{Var}[s_{ij}] = \mathbf{E}[s_{ij}^2] - (\mathbf{E}[s_{ij}])^2 \leq \mathbf{E}[s_{ij}^2].$$

We next show $\mathbf{E}[s_{ij}^2]$ is bounded by $4B^2$:

$$\begin{aligned}
\mathbf{E}[s_{ij}^2] &= \mathbf{E}\left[(z_i - \hat{\mu}_i)^2 (z_j - \hat{\mu}_j)^2\right] \\
&\leq \sqrt{\mathbf{E}\left[(z_i - \hat{\mu}_i)^4\right]} \sqrt{\mathbf{E}\left[(z_j - \hat{\mu}_j)^4\right]} \\
&\leq \sqrt{(\mu_i - \hat{\mu}_i)^4 + 6(\mu_i - \hat{\mu}_i)^2 \sigma_{ii} + 3\sigma_{ii}^2} \cdot \sqrt{(\mu_j - \hat{\mu}_j)^4 + 6(\mu_j - \hat{\mu}_j)^2 \sigma_{jj} + 3\sigma_{jj}^2} \\
&\leq \epsilon^4 + 6\epsilon^2 B + 3B^2 \\
&\leq 4B^2,
\end{aligned}$$

where the first step is by definition, the second by the Cauchy-Schwarz inequality (Lemma 67), the third due to Lemma 59, the fourth due to our assumption that $|\hat{\mu}_i - \mu_i| \leq \epsilon$, $|\hat{\mu}_j - \mu_j| \leq \epsilon$, and $\max_{ij} \sigma_{ij} \leq B$, and the last is because $\epsilon < 1/3$ and $B \geq 1$. □

6.7.4 Proof of Lemma 19

We first derive an upper bound of the total variation between two multivariate normal distributions:

$$\begin{aligned}
&\mathrm{d_{var}}\left(\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\Sigma})\right) \\
&\quad\leq \mathrm{d_{var}}\left(\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\hat{\boldsymbol{\mu}}, \Sigma)\right) + \mathrm{d_{var}}\left(\mathcal{N}(\hat{\boldsymbol{\mu}}, \Sigma), \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\Sigma})\right) \\
&\quad\leq \sqrt{2\mathrm{d_{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\hat{\boldsymbol{\mu}}, \Sigma)\right)} + \sqrt{2\mathrm{d_{KL}}\left(\mathcal{N}(\hat{\boldsymbol{\mu}}, \Sigma), \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\Sigma})\right)} \\
&\quad= \sqrt{(\boldsymbol{\mu} - \hat{\boldsymbol{\mu}})^\top \Sigma^{-1} (\boldsymbol{\mu} - \hat{\boldsymbol{\mu}})} + \sqrt{\ln \frac{\det \hat{\Sigma}}{\det \Sigma} + \mathrm{tr}\left(\hat{\Sigma}^\top \Sigma\right) - n},
\end{aligned} \tag{6.6}$$

where the first step is due to the triangle inequality, the second uses Lemma 58, and the last uses Lemma 62. The rest of our proof relies on bounding each term of the right-hand side above.

To simplify the final expression, we will bound some characteristics of the covariance matrix Σ. Let $\lambda_1 \geq \ldots \geq \lambda_n > 0$ be the singular values of the non-singular covariance matrix Σ, and Σ^{-1} its inverse. Since Σ is symmetric and positive definite, λ_is are also its eigenvalues. Also recall that we assume Σ is bounded by $-B$ and B component-wise.

Lemma 28 *Using the notation and assumption above, we have:*

$$\begin{aligned} \operatorname{tr}(\Sigma) &\leq nB \\ \lambda_n &\leq \max_i \sigma_{ii} \leq B \\ \left\|\Sigma^{-1}\right\|_1 &\leq \sqrt{n}\left\|\Sigma^{-1}\right\|_2 = \frac{\sqrt{n}}{\lambda_n}. \end{aligned}$$

PROOF. We prove the three upper bounds one by one:

1. The first follows from the fact that every element in Σ is between $-B$ and B.

2. The second follows almost immediately from Lemma 64:

$$\lambda_n \leq \frac{1}{n}\sum_{i=1}^{n}\lambda_i = \frac{\operatorname{tr}(\Sigma)}{n} \leq B.$$

3. It is known that $\|A\|_1 \leq \sqrt{n}\,\|A\|_2$ for any $n \times n$ matrix A (see, *e.g.*, Theorem 5.6.18 of Horn and Johnson [1986]). On the other hand, $\left\|\Sigma^{-1}\right\|_2$ equals the largest singular value of Σ^{-1}, which is $1/\lambda_n$.

□

We now start with bounding the first term in Equation 6.6:

Lemma 29 *Using the notation and assumptions above, we have:*

$$(\boldsymbol{\mu} - \hat{\boldsymbol{\mu}})^\top \Sigma^{-1}(\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}) \leq \frac{1}{\lambda_n}\|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}\|_2^2.$$

PROOF. First note that, since Σ^{-1} is symmetric, the ratio

$$\frac{(\boldsymbol{\mu} - \hat{\boldsymbol{\mu}})^\top \Sigma^{-1}(\boldsymbol{\mu} - \hat{\boldsymbol{\mu}})}{\|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}\|_2^2}$$

is a *Rayleigh quotient*, which is upper-bounded by the largest singular value of Σ^{-1}, which is λ_n^{-1}. □

We then move to bounding the second term in Equation 6.6:

Lemma 30 *Using the notation and assumptions above, if* $\max_{i,j}|\sigma_{ij} - \hat{\sigma}_{ij}| \leq \epsilon$, *then*

$$\ln\frac{\det\hat{\Sigma}}{\det\Sigma} \leq n\epsilon\left(\frac{1}{\lambda_1} + \frac{1}{\lambda_2} + \cdots + \frac{1}{\lambda_n}\right) \leq \frac{n^2\epsilon}{\lambda_n}.$$

PROOF. Define $E = [e_{ij}]_{ij} = \hat{\Sigma} - \Sigma$. Clearly, E is symmetric since both Σ and $\hat{\Sigma}$ are. Its eigenvalues are denoted by $\psi_1 \geq \psi_2 \geq \cdots \geq \psi_n$, which are real (but can be negative or positive or zero). According to Lemma 64,

$$\ln \frac{\det \hat{\Sigma}}{\det \Sigma} = \ln \prod_{i=1}^{n} \frac{\hat{\lambda}_i}{\lambda_i} = \sum_{i=1}^{n} \ln \frac{\hat{\lambda}_i}{\lambda_i}.$$

By Geršgorin's theorem [Horn and Johnson 1986, Theorem 6.1.1], the eigenvalues of E must be small as the elements of E are small. Specifically, any ψ_i must lie in one of the n *Geršgorin discs* D_j ($j = 1, 2, \ldots, n$):

$$D_j \stackrel{\text{def}}{=} \left\{ x \in \mathbb{R} \mid |x - e_{jj}| \leq \sum_{j' \neq j} \left| e_{jj'} \right| \right\}.$$

It follows for all i that

$$|\psi_i| \leq \max_j \sum_{j'=1}^{n} \left| e_{jj'} \right| \leq n\epsilon$$

as every component in E lies in the range $[-\epsilon, \epsilon]$.

On the other hand, from Weyl's theorem [Horn and Johnson 1986, Theorem 4.3.1], we have

$$\psi_1 \geq \hat{\lambda}_i - \lambda_i \geq \psi_n.$$

We have just proved that both $|\psi_1|$ and $|\psi_n|$ are at most $n\epsilon$, and thus

$$\left| \hat{\lambda}_i - \lambda_i \right| \leq n\epsilon.$$

Consequently,

$$\frac{\hat{\lambda}_i}{\lambda_i} \leq \frac{\lambda_i + n\epsilon}{\lambda_i} = 1 + \frac{n\epsilon}{\lambda_i}.$$

Finally, by putting all pieces together, we have

$$\ln \frac{\det \hat{\Sigma}}{\det \Sigma} = \sum_{i=1}^{n} \ln \frac{\hat{\lambda}_i}{\lambda_i} \leq \sum_{i=1}^{n} \ln \left(1 + \frac{n\epsilon}{\lambda_i} \right) \leq \sum_{i=1}^{n} \frac{n\epsilon}{\lambda_i} \leq \frac{n^2 \epsilon}{\lambda_n},$$

where the second-to-last inequality uses the inequality $\ln(1 + x) \leq x$ for $x \geq 0$. □

Now, we bound the third term of Equation 6.6:

Lemma 31 *Using the notation and assumption above, if* $\max_{i,j} |\sigma_{ij} - \hat{\sigma}_{ij}| \leq \epsilon$ *and* $n\epsilon \left\| \Sigma^{-1} \right\|_1 < 1$, *then*

$$\operatorname{tr}\left(\hat{\Sigma}^{-1} \Sigma \right) - n \leq \frac{2n^3 \epsilon B}{\lambda_n^2 - n^{3/2} \lambda_n \epsilon}.$$

Note that $\epsilon \leq \lambda_n n^{-3/2}$ *suffices to guarantee* $n\epsilon \left\| \Sigma^{-1} \right\|_1 < 1$ *due to Lemma 28.*

PROOF. The i-th row (or column) of Σ^{-1} is the solution to the system of linear equations: $\Sigma\mathbf{z} = \mathbf{e}_i$ where $\mathbf{e}_i$ has $n-1$ zero components except a 1 in the i-th component. Similarly, the i-th row (or column) of $\hat{\Sigma}^{-1}$ is the solution to $\hat{\Sigma}\hat{\mathbf{z}} = \mathbf{e}_i$. Since Σ and $\hat{\Sigma}$ differ by at most ϵ in every component, we have

$$\frac{\left\|\Sigma - \hat{\Sigma}\right\|_1}{\|\Sigma\|_1} \leq \frac{n\epsilon}{\|\Sigma\|_1}.$$

For convenience, denote the right-hand side by ϵ'. It follows from a standard perturbation result in numerical analysis (Lemma 65) that

$$\|\mathbf{z} - \hat{\mathbf{z}}\|_1 \leq \frac{2\epsilon'\kappa_1(\Sigma)\,\|\mathbf{z}\|_1}{1 - \epsilon'\kappa_1(\Sigma)},$$

where the condition number $\kappa_1(\cdot)$ is defined by

$$\kappa_1(\Sigma) \stackrel{\text{def}}{=} \|\Sigma\|_1 \left\|\Sigma^{-1}\right\|_1.$$

The above inequality holds for all n possible $\mathbf{e}_i$ vectors. Note that $\|\mathbf{z} - \hat{\mathbf{z}}\|_1$ is the absolute sum of the i-th row (or column) of $\Sigma^{-1} - \hat{\Sigma}^{-1}$ if $\mathbf{e}_i$ is used in the system of linear equations. Let $\psi_1 \geq \psi_2 \geq \cdots \geq \psi_N \geq 0$ be the singular values of $\Sigma^{-1} - \hat{\Sigma}^{-1}$. These singular values are in general different from the eigenvalues of $\Sigma^{-1} - \hat{\Sigma}^{-1}$, but they coincide with the *absolute values* of the eigenvalues of the symmetric matrix $\Sigma^{-1} - \hat{\Sigma}^{-1}$. It follows from this fact, the interpretation of $\|\mathbf{z}\|_1$ above, and Geršgorin's theorem that for all i,

$$\psi_i \leq \max_{\mathbf{e}_i} \|\mathbf{z} - \hat{\mathbf{z}}\|_1 \leq \frac{2\epsilon'\kappa_1(\Sigma)}{1 - \epsilon'\kappa_1(\Sigma)} \max_{\mathbf{e}_i} \|\mathbf{z}\|_1 = \frac{2\epsilon'\kappa_1(\Sigma)}{1 - \epsilon'\kappa_1(\Sigma)} \left\|\Sigma^{-1}\right\|_1.$$

We can now finish the proof:

$$\begin{aligned}
\operatorname{tr}\left(\hat{\Sigma}^{-1}\Sigma\right) - n \;&=\; \operatorname{tr}\left((\hat{\Sigma}^{-1} - \Sigma^{-1})\Sigma\right) \\
&\leq \sum_{i=1}^{n} \psi_i \lambda_i \\
&\leq \frac{2\epsilon' \kappa_1(\Sigma) \left\|\Sigma^{-1}\right\|_1}{1 - \epsilon' \kappa_1(\Sigma)} \sum_{i=1}^{n} \lambda_i \\
&= \frac{2\epsilon' \kappa_1(\Sigma) \left\|\Sigma^{-1}\right\|_1}{1 - \epsilon' \kappa_1(\Sigma)} \operatorname{tr}(\Sigma) \\
&= \frac{2n\epsilon \left\|\Sigma^{-1}\right\|_1^2}{1 - n\epsilon \left\|\Sigma^{-1}\right\|_1} \operatorname{tr}(\Sigma) \\
&\leq \frac{2n^2 B\epsilon \left\|\Sigma^{-1}\right\|_1^2}{1 - n\epsilon \left\|\Sigma^{-1}\right\|_1}, \\
&\leq \frac{2n^3 \epsilon B}{\lambda_n^2 - n^{3/2}\lambda_n \epsilon},
\end{aligned}$$

where the first equality is due to the identity $\operatorname{tr}\left(\Sigma^{-1}\Sigma\right) = \operatorname{tr}(I_n) = n$, the first inequality is a direct application of von Neumann's inequality (Lemma 66), the second inequality makes use of the upper bound for ψ_i that we have proved earlier, the second equality is due to Lemma 64, and the third equality is obtained by the definitions of ϵ' and $\kappa_1(\Sigma)$, and the last two inequalities are due to the upper bounds on $\operatorname{tr}(\Sigma)$ and $\left\|\Sigma^{-1}\right\|_1$ provided in Lemma 28. □

Combining Equation 6.6 with Lemmas 29–31, we can complete the proof.

Part III

PAC-MDP Reinforcement Learning

Chapter 7

Model-based Approaches

This chapter studies model-based PAC-MDP algorithms. A new algorithm called KWIK-Rmax is proposed that unifies existing PAC-MDP algorithms in the literature. Novel PAC-MDP algorithms are also developed using tools from the KWIK learning framework. As case studies, we present two experiments showing KWIK-Rmax is effective in practice, in addition to its strong theoretical guarantee.

7.1 A Generic Model-based PAC-MDP Algorithm

Recall that a model-based RL algorithm often learns a model of the unknown MDP and then computes an optimal policy according to the learned model. These algorithms are often considered more sample efficient. In fact, the earliest PAC-MDP algorithms are both model based [Kearns and Singh 2002; Brafman and Tennenholtz 2002]. At the core of all existing model-based PAC-MDP algorithms lies the idea of distinguishing between *known states*—states where the transition distribution and rewards can be accurately inferred from observed transitions—and *unknown states*.

As an example, consider the Rmax algorithm (Algorithm 24) for finite MDPs. At the beginning of learning, the algorithm initializes its internal MDP model, $\hat{M}$, to an optimistic one that assigns $V_{\max}$ value to all states. It then follows an optimal policy in $\hat{M}$ to collect observations about the true MDP's dynamics through the standard online-interaction protocol. When some state–action pair, (s, a), has been experienced m times, the algorithm updates its internal model using maximum-likelihood estimates from observations for that pair. Now, observe that the transition probabilities, $T(\cdot \mid s, a)$, and reward function, $R(s, a)$, can be learned using a small number of samples to within arbitrary precision with high probability, using tools developed in §5. Therefore,

as long as m is large enough, the estimated transition and function functions in (s, a) are accurate. Furthermore, we may guarantee that Rmax either explores (by visiting an unknown state–action pair) or exploits (by following a near-optimal policy in the true MDP), and the number of exploration steps is bounded by a polynomial, as shown by Kakade [2003]. These results together show that Rmax is PAC-MDP.

Algorithm 24 Rmax

Inputs: $\mathcal{S}, \mathcal{A}, \gamma, \epsilon, \delta, m$.
Initialize counter $c(s, a) \leftarrow 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$.
Initialize the empirical known state–action MDP $\hat{M} = \langle \mathcal{S}, \mathcal{A}, \hat{T}, \hat{R}, \gamma \rangle$:

$$\hat{T}(s' \mid s, a) = \mathbb{I}(s' = s), \qquad \hat{R}(s, a) = V_{\max}(1 - \gamma).$$

for all timesteps $t = 1, 2, 3, \ldots$ **do**
  Compute an optimal policy π_t of $\hat{M}$ using any exact planning algorithms in §3.1.
  Observe the current state s_t, take action $a_t = \pi_t(s_t)$, receive reward r_t, and transition to the next state s_{t+1}.
  $c(s_t, a_t) \leftarrow c(s_t, a_t) + 1$
  if $c(s_t, a_t) = m$ **then**
    Change $\hat{T}(\cdot \mid s_t, a_t)$ and $\hat{R}(s_t, a_t)$ using maximum-likelihood estimates (*c.f.*, Equations 4.2 and 4.3) based on the m observed transitions.
  end if
end for

A key observation, which is the focus of this section, is that, if a class of MDPs can be KWIK-learned, then there exists an Rmax-style algorithm that is PAC-MDP for this class of MDPs. This idea is formalized in Theorem 21, which is proved through construction of the KWIK-Rmax algorithm (Algorithm 25). The proof relies on a form of the *simulation lemma* (Lemma 33), which relates value-function approximation error to model approximation error, and on a generic PAC-MDP theorem in §4.4.4.

Definition 13 *Fix the state space $\mathcal{S}$, action space $\mathcal{A}$, and discount factor γ.*

1. *Define $\mathcal{X} = \mathcal{S} \times \mathcal{A}$, $\mathcal{Y}_T \subseteq \mathcal{P}_{\mathcal{S}}$, and $\mathcal{Z}_T = \mathcal{S}$. Let $\mathcal{H}_T \subseteq \mathcal{X}^{\mathcal{Y}_T}$ be the set of transition functions of an MDP. $\mathcal{H}_T$ is* (efficiently) KWIK-learnable *(when the observation set is $\mathcal{Z}_T$) if in the accuracy requirement of Definition 11, $\left|\hat{T}(\cdot \mid s, a) - T(\cdot \mid s, a)\right|$*

is interpreted as the ℓ_1-distance:

$$\left|\hat{T}(\cdot \mid s,a) - T(\cdot \mid s,a)\right| \stackrel{\text{def}}{=} \begin{cases} \sum_{s' \in \mathcal{S}} \left|\hat{T}(s' \mid s,a) - T(s' \mid s,a)\right| & \text{if } \mathcal{S} \text{ is countable} \\ \int_{s' \in \mathcal{S}} \left|\hat{T}(s' \mid s,a) - T(s' \mid s,a)\right| ds' & \text{otherwise.} \end{cases}$$

2. *Define $\mathcal{X} = \mathcal{S} \times \mathcal{A}$ and $\mathcal{Y}_R = \mathcal{Z}_R = [0,1]$. Let $\mathcal{H}_R \subseteq (\mathcal{Y}_R)^{\mathcal{X}}$ be the set of reward functions of an MDP. $\mathcal{H}_R$ is* (efficiently) KWIK-learnable *(when the observation set is $\mathcal{Z}_R$) if in the accuracy requirement of Definition 11, $|\hat{R}(s,a) - R(s,a)|$ is interpreted as the absolute value.*

3. *Let $\mathcal{M} = \{M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle \mid T \in \mathcal{H}_T, R \in \mathcal{H}_R\}$ be a class of MDPs. $\mathcal{M}$ is* (efficiently) KWIK-learnable *if both $\mathcal{H}_T$ and $\mathcal{H}_R$ are (efficiently) KWIK-learnable.*

Theorem 21 *Let $\mathcal{M}$ be a class of MDPs with state space $\mathcal{S}$ and action space $\mathcal{A}$. If $\mathcal{M}$ can be (efficiently) KWIK-learned by algorithms $\mathbf{A}_T$ (for transition functions) and $\mathbf{A}_R$ (for reward functions), then* KWIK-Rmax *is PAC-MDP. In particular, if the following parameters are used,*

$$\epsilon_T = \Theta(\epsilon(1-\gamma)^2), \quad \epsilon_R = \Theta(\epsilon(1-\gamma)), \quad \epsilon_P = \Theta(\epsilon(1-\gamma)), \quad \delta_T = \delta_R = \Theta(\delta),$$

then the sample complexity of exploration of KWIK-Rmax *is*

$$\mathcal{O}\left(\frac{V_{\max}}{\epsilon(1-\gamma)}\left(B_T\left(\epsilon(1-\gamma)/V_{\max}, \delta\right) + B_R(\epsilon(1-\gamma), \delta) + \ln\frac{1}{\delta}\right)\ln\frac{1}{\epsilon(1-\gamma)}\right). \quad (7.1)$$

The algorithm KWIK-Rmax (Algorithm 25) relies on two KWIK algorithms, $\mathbf{A}_T$ (using parameters ϵ_T, δ_T) and $\mathbf{A}_R$ (using parameters ϵ_R, δ_R), for KWIK-learning the MDP's transition and reward functions, respectively, and maintains an estimate of the MDP called the *empirical known state–action MDP*. The estimate distinguishes two types of state–actions: for state–actions where both $\mathbf{A}_T$ and $\mathbf{A}_R$ can make valid predictions, the predictions must be accurate with high probability (thanks to the accuracy requirement for KWIK algorithms) and thus their dynamics are *known*; for other state–actions, their transition and reward functions cannot be accurately estimated and thus they are *unknown*. By assigning the largest possible value (which is $V_{\max}$) to all unknown state–actions, the agent is encouraged to explore these state–actions, unless the

Algorithm 25 KWIK-Rmax

Inputs: $\mathcal{S}, \mathcal{A}, \gamma, \mathbf{A}_T, \mathbf{A}_R, \epsilon_T, \delta_R, \epsilon_R, \delta_R, \epsilon_P$.
Run $\mathbf{A}_T$ with parameters ϵ_T and δ_T.
Run $\mathbf{A}_R$ with parameters ϵ_R and δ_R.
for all timesteps $t = 1, 2, 3, \ldots$ **do**
  Update the empirical known state–action MDP $\hat{M} = \langle \mathcal{S}, \mathcal{A}, \hat{T}, \hat{R}, \gamma \rangle$
  for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ **do**
    if $\mathbf{A}_T(s, a) = \perp$ or $\mathbf{A}_R(s, a) = \perp$ **then**
      $\hat{T}(s' \mid s, a) = \begin{cases} 1 & \text{if } s' = s \\ 0 & \text{otherwise} \end{cases}$ and $\hat{R}(s, a) = V_{\max}(1 - \gamma)$.
    else
      $\hat{T}(\cdot \mid s, a) = \mathbf{A}_T(s, a)$ and $\hat{R}(s, a) = \mathbf{A}_R(s, a)$.
    end if
  end for
  Compute a near-optimal value function Q_t of $\hat{M}$ such that $\left| Q_t(s, a) - Q^*_{\hat{M}}(s, a) \right| \leq \epsilon_P$ for all (s, a), where $Q^*_{\hat{M}}$ is the optimal state–action value function of $\hat{M}$.
  Observe the current state s_t, take action $a_t = \arg\max_{a \in \mathcal{A}} Q_t(s_t, a)$, receive reward r_t, and transition to the next state s_{t+1}.
  if $\mathbf{A}_T(s_t, a_t) = \perp$ **then**
    Inform $\mathbf{A}_T$ of the sample $(s_t, a_t) \to s_{t+1}$.
  end if
  if $\mathbf{A}_R(s_t, a_t) = \perp$ **then**
    Inform $\mathbf{A}_R$ of the sample $s_t \to r_t$.
  end if
end for

probability of reaching them is too small, in which case the policy is near-optimal. Since the number of visits to an unknown state–action is polynomial in the relevant quantities (the sample complexity requirement for KWIK algorithms), the number of timesteps the algorithm does not behave near-optimally is also a polynomial. This intuition is exactly the basic idea of our PAC-MDP proof in §7.4.1.

A few caveats are in order regarding practical issues when instantiating and implementing Algorithm 25:

1. The definitions of $\hat{T}$ and $\hat{R}$ given in Algorithm 25 are *conceptual* rather than *operational*. For finite MDPs, one may represent $\hat{T}$ by a matrix of size $\mathcal{O}(|\mathcal{S}|^2 |A|)$ and $\hat{R}$ by a vector of size $\mathcal{O}(|\mathcal{S}| |\mathcal{A}|)$. For structured MDPs, more compact representations are possible. For instance, MDPs for some linear dynamical systems can be represented by matrices of finite dimension (§7.2.2 and §7.2.3), and factored

MDPs can be represented by a dynamic Bayes net (§7.2.4 and §7.2.5).

2. It is unnecessary to update $\hat{T}$ and $\hat{R}$ and recompute Q_t for every timestep t. The known state–action MDP $\hat{M}$ (and thus $Q^*_{\hat{M}}$ and Q_t) remains unchanged unless some unknown state–action becomes known. Therefore, one may update $\hat{M}$ and Q_t only when $\mathbf{A}_T$ or $\mathbf{A}_R$ obtain new samples in lines 13 or 16.

3. It is unnecessary to compute $Q^*_{\hat{M}}$ for all (s, a). In fact, it suffices to guarantee that Q_t is ϵ_P-accurate in state s_t: $\left|Q_t(s_t, a) - Q^*_{\hat{M}}(s_t, a)\right| < \epsilon_P$ for all $a \in \mathcal{A}$. This kind of *local* planning often requires significantly less computation than *global* planning (*c.f.*, §3.3). However, to make use of Theorem 4, it has to be guaranteed that Q_t changes at most a polynomial number of times.

4. Given the approximate MDP $\hat{M}$ and the current state s_t, the algorithm computes a near-optimal action for s_t. This step can be done efficiently using dynamic programming for finite MDPs. In general, however, doing so is computationally expensive [Chow and Tsitsiklis 1989]. Fortunately, recent advances in approximate local planning have made it possible for large-scale problems [Kearns et al. 2002; Kocsis and Szepesvári 2006].

The remaining sections consider various subclasses of MDPs and unify most existing PAC-MDP algorithms using the KWIK framework developed in previous sections. §7.2.1 considers finite MDPs without considering generalization across states, while §§7.2.2—7.2.4 show how generalization can be combined with KWIK to make use of structural assumptions of corresponding MDP classes to obtain more efficient RL algorithms. Finally, §7.2.5 shows how we may obtain a PAC-MDP algorithm for factored-state MDPs with unknown factorization structures by streamlining four KWIK algorithms we have developed, resulting in a novel algorithm that is significantly better than the state-of-the-art result. In all these cases, we focus on learning the transition function and assume the reward function is known. The extension to KWIK-learning reward functions is straightforward.

7.2 KWIK-Learnable MDP Classes

In this section, we examine several classes of MDPs and show they can be KWIK-learned by techniques studied in Part II. The KWIK bounds and relevant KWIK algorithms are summarized in Table 7.1. In light of Theorem 21, these KWIK bounds imply corresponding sample complexity of exploration bounds for KWIK-Rmax running on these MDPs. We also note a few others KWIK-learnable MDPs which are not covered in detail here, including [Diuk et al. 2008] [Asmuth et al. 2008] [Leffler et al. 2007] [Walsh et al. 2009b].

Case	**Subalgorithms**	**KWIK Bound**
Finite	input-partition ← dice-learning	$\mathcal{O}\left(\frac{n^2 m}{\epsilon^2} \ln \frac{nm}{\delta}\right)$
Linear Dynamics	output-combination ← noisy linear-regression	$\tilde{\mathcal{O}}\left(\frac{n_S^2 n}{\epsilon^4}\right)$
Normal Offset	input-partition ← output-combination ← coin-learning	$\mathcal{O}\left(\frac{n^8 n_\tau^2 m^2}{\lambda_{\min}^2 \epsilon^4 \delta}\right)$
Factored with Known $\mathcal{P}$	cross-product ← input-partition ← dice-learning	$\mathcal{O}\left(\frac{n^3 m D N^{D+1}}{\epsilon^2} \ln \frac{nmN}{\delta}\right)$
Factored with Unknown $\mathcal{P}$	output-combination ← noisy union ← input-partition ← dice-learning	$\mathcal{O}\left(\frac{n^{D+3} m D N^{D+1}}{\epsilon^2} \ln \frac{nmN}{\delta}\right)$

Table 7.1: KWIK bounds and the component KWIK algorithms for a number of prominent MDP classes. Refer to the text for definitions of the quantities in the KWIK bounds.

7.2.1 Finite MDPs

Finite MDPs have received the bulk of the attention in the literature because of their mathematical simplicity and generality. A finite MDP $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ consists of $n = |\mathcal{S}|$ states and $m = |\mathcal{A}|$ actions. For each combination of state (s) and action (a) and next state (s'), the transition function returns a probability, denoted $T(s' \mid s, a)$. As the reinforcement-learning agent moves around in the state space, it observes state–action–next-state transitions and must predict the probabilities for transitions it has not yet observed. In the model-based setting, an algorithm learns a mapping from the size nm input set of state–action combinations to multinomial distributions over the next states via multinomial observations. Thus, the problem of learning the model can be solved via input-partition over a set of individual probabilities learned via dice-learning. The resulting KWIK bound is:

$$\begin{aligned} B_{\text{finite MDP}}(\epsilon, \delta) &= \sum_{(s,a) \in \mathcal{S} \times \mathcal{A}} B_{\text{dice-learning}}\left(\epsilon, \frac{\delta}{nm}\right) \\ &= nm\, \mathcal{O}\left(\frac{n}{\epsilon^2} \ln \frac{n^2 m}{\delta}\right) \\ &= \mathcal{O}\left(\frac{n^2 m}{\epsilon^2} \ln \frac{nm}{\delta}\right), \end{aligned}$$

where the KWIK bounds of input-partition and dice-learning are used in the first two equalities.

This approach is precisely what is found in almost all sample-efficient RL algorithms in the literature for finite MDPs [Kearns and Singh 2002; Brafman and Tennenholtz 2002; Kakade 2003; Strehl et al. 2006a]. Applying Theorem 21 with the KWIK bound $B_{\text{finite MDP}}$, we recover essentially the same sample complexity of exploration found by Kakade [2003]:[1]

$$\mathcal{O}\left(\frac{n^2 m {V_{\max}}^3}{\epsilon^3 (1-\gamma)^3} \ln \frac{nm}{\delta} \ln \frac{1}{\epsilon(1-\gamma)}\right) = \tilde{\mathcal{O}}\left(\frac{n^2 m {V_{\max}}^3}{\epsilon^3 (1-\gamma)^3}\right).$$

A few notes are in places regarding extensions of the two-level algorithm for KWIK-learning finite MDPs above:

[1] Our bound is tighter than the bound by Kakade [2003] due to a slightly more careful analysis. But the basic idea of the proofs are the same.

- In many problems, taking an action in a state can often land the agent to a small number of next states. Classic examples are the various gridworlds [Sutton and Barto 1998], in which there are at most five possible next states from any (s, a) pair: s itself, and the states to the north, east, south, and west of s. Formally, we may assume the number of next states from any state–action pair is upper bounded by some number $k \in \mathbb{N}$:

$$k \geq \max_{(s,a) \in \mathcal{S} \times \mathcal{A}} \left| \{ s' \in \mathcal{S} \mid T(s' \mid s, a) > 0 \} \right|,$$

  and $k \ll n$. In such MDPs, we may use the same algorithm above, but the KWIK bound for each dice-learning component can be tightened, yielding an improved KWIK bound of

$$\mathcal{O}\left(\frac{nmk}{\epsilon^2} \ln \frac{nm}{\delta} \right).$$

  This bound is linearithmic in n when k is a constant.

- A similar algorithm can KWIK-learn an MDP with an infinite state space, provided that the MDP satisfy a *local modeling assumption* [Kakade et al. 2003]. Roughly speaking, this assumption requires the following: for any (s, a) pair, if we have k IID samples, $\mathcal{D} = \{(s_1, a, r_1, s'_1), \ldots, (s_k, a, r_k, s'_k)\}$, such that every s_i is in a small enough neighborhood of s, then we can accurately infer the unknown dynamics, $R(s, a)$ and $T(\cdot \mid s, a)$, from $\mathcal{D}$ with high confidence. The KWIK bound for learning this type of MDP is in general (at least) linear in a quantity known as the *covering number*, which may be finite and much less than $|\mathcal{S}|$.

- For an MDP with intractably large a state space, if there is a model-irrelevance abstraction (§3.2.1) such that the corresponding abstract MDP has finitely many abstract states, then we can apply the two-level algorithm to KWIK-learn the abstract MDP. The resulting KWIK bound depends on $\left|\bar{\mathcal{S}}\right|$ instead of $|\mathcal{S}|$.

- Finally, interested readers are referred to a similar analysis in finite MDPs where the outcomes of a transition, including the reward and next state, are not observed by the agent immediately, but will be delayed for a constant timesteps K [Walsh et al. 2007; 2009a]. Such *delayed-observation MDPs* can be turned into a regular MDP, called the "augmented MDP", with exponentially (in K)

many states [Katsikopoulos and Engelbrecht 2003]. Fortunately, we may exploit a special structure (thanks to the constant observation delay) in the augmented MDP's dynamics, and to learn the MDP with a polynomial KWIK bound.

Before closing this subsection, it is appropriate to mention an algorithm similar to Rmax (namely, KWIK-Rmax in finite MDPs), which uses an incremental planning algorithm to solve the known-state MDP and thus enjoys a much lower computational complexity. The algorithm, RTDP-Rmax [Strehl et al. 2006a], maintains a known-state MDP like Rmax, but then uses RTDP (§3.1.2) to refine its Q-function incrementally. Surprisingly, this algorithm's sample complexity of exploration is the same as Rmax's (ignoring logarithmic factors), despite the laziness nature of its planning component. Two related algorithms are analyzed in §8.1.

In a real-time system, an RL agent may have to decide what action to take in a given time period. This time period may be too short to solve the known-state MDP completely (as in Rmax), but may be large enough to afford more than one Bellman backup (as in RTDP-Rmax). A natural idea is for the agent to perform as many Bellman backups as possible within the given time period. Motivated by this observation, Strehl [2007b] combines prioritized sweeping with RTDP-Rmax and argue that the resulting algorithm, which we call PS-Rmax, remains PAC-MDP. PS-Rmax contains RTDP-Rmax as a special case in which only one Bellman backup is allowed per timestep. On the other extreme, the asymptotic convergence results established in §3.1.2 imply that PS-Rmax approaches Rmax when more and more Bellman backups are affordable in each timestep.

7.2.2 MDPs with Linear Dynamics

In many robotics and control applications, the systems being manipulated possess infinite state spaces and action spaces, but their dynamics are governed by a system of linear equations (see, *e.g.*, Sontag [1998]). Our model formulation is based on Strehl and Littman [2008b], and is similar to Abbeel and Ng [2005]. Here, $\mathcal{S} \subseteq \mathbb{R}^{n_S}$ and $\mathcal{A} \subseteq \mathbb{R}^{n_A}$ are the state and action spaces, respectively. The transition function T is a

multivariate normal distribution:

$$T(\cdot \mid s, a) = \mathcal{N}\left(F\boldsymbol{\phi}(s,a), \sigma^2 I\right),$$

where $\boldsymbol{\phi} \in (\mathbb{R}^n)^{\mathbb{R}^{n_S+n_A}}$ is a basis function satisfying $\|\boldsymbol{\phi}(s,a)\|_2 \leq 1$ for all (s,a), $F \in \mathbb{R}^{n_S \times n}$ is a fixed but unknown matrix, σ^2 is some positive number, and $I \in \mathbb{R}^{n_S \times n_S}$ is the identity matrix. We assume $\boldsymbol{\phi}$ and σ^2 are given, but F is unknown.

For such linearly parameterized transition functions, the expectation of each component of the next state is linear in the feature of the current state–action pair, and the coefficients correspond to the numbers in the corresponding rows in F. To guarantee ϵ_T ℓ_1-distance between $\hat{T}$ and T as required by Theorem 21, it follows from Proposition 7 of Abbeel and Ng [2005] that we only need to guarantee the mean vector, $F\boldsymbol{\phi}(s,a)$, is predicted to within $\sqrt{\pi/2}\sigma\epsilon_T$ in the ℓ_2-norm. Now the transition function of the class of linearly parameterized MDPs can be KWIK-learned via output-combination over the n_S state components using $\alpha = 1/\sqrt{n_S}$, each of which can be KWIK-learned by noisy linear-regression. The resulting KWIK bound is

$$B_{\text{linear MDP}}(\epsilon, \delta) \;=\; B_{\text{noisy linear-regression}}\left(\frac{\sqrt{\pi/2}\sigma\epsilon}{\sqrt{n_S}}, \frac{\delta}{n_S}\right) \;=\; \tilde{\mathcal{O}}\left(\frac{n_S^2 n}{\sigma^4\epsilon^4}\right),$$

where the first step applies Proposition 7 of Abbeel and Ng [2005] and the KWIK bound of output-combination, and the second uses the recent result of Walsh et al. [2009b]. Combining $B_{\text{liner MDP}}$ with Theorem 21, we obtain the following sample complexity of exploration upper bound of KWIK-Rmax in linear MDPs:

$$\tilde{\mathcal{O}}\left(\frac{n_S^2 n {V_{\max}}^5}{\sigma^4\epsilon^5(1-\gamma)^5}\right).$$

7.2.3 Typed MDPs with Normal Offset Dynamics

The linear transition model above is a class of parameterized transition functions for continuous MDPs. A related class is called *typed dynamics* (see, *e.g.*, Leffler et al. [2007]), where the transition functions depends on a *type* $\tau(s)$ assigned to each state s and action a.

Here, we consider a specific subset of this class, adopted from Brunskill et al. [2008], where $\mathcal{S} \subseteq \mathbb{R}^n$ and the next state distribution is given by the following multivariate

normal distribution:

$$T(\cdot \mid s, a) \sim \mathcal{N}\left(s + \mu_{\tau(s)a}, \Sigma_{\tau(s)a}\right),$$

where $\mu_{\tau(s)a} \in \mathbb{R}^n$ and $\Sigma_{\tau(s)a} \in \mathbb{R}^{n \times n}$ are the mean and covariance matrix of the normal distribution for the type–action pair $(\tau(s), a)$. In other words, the *offset* of states is normally distributed. We assume the number of actions, denoted $m = |\mathcal{A}|$, and the number of types, denoted $n_\tau = |\{\tau(s) \mid s \in \mathcal{S}\}|$, are both finite.

An example domain where such dynamics may arise is robot navigation across varying terrain. The distribution of the offset of the robot's position after taking an action (such as turn-left or go-forward) depends on this action as well as the type of the terrain (such as sand, wood, or ice, etc.). In many real-world applications, the number of types is often small although the number of states can be astronomically large or even infinite. Typed offset dynamics, therefore, provide a compact way to represent the MDP. A robot navigation example is considered in §7.3.2, while more motivations and experiments are found in Brunskill et al. [2009].

For each type–action pair (τ, a), the components in $\mu_{\tau a}$ and $\Sigma_{\tau a}$ can be interpreted as means of observations. For instance, assume that we have acquired a sample transition, (s, a, r, s'), then by definition

$$\mu_{\tau(s)a}[i] = \mathbf{E}_{s' \sim T(\cdot|s,a)}\left[s'[i] - s[i]\right]$$

for all $i = 1, 2, \ldots, n$. Therefore, we may decompose the problem of KWIK-learning the offset means using input-partition over all type–action pairs. For each (τ, a) pair, learning the corresponding normal distribution can be done by Algorithm 23. The KWIK bound for learning typed, normally offset MDPs is thus

$$B_{\text{typed, normal-offset MDP}}(\epsilon, \delta) = \sum_{(\tau,a)} B_{\text{normal learning}}\left(\epsilon, \frac{\delta}{n_\tau m}\right) = \mathcal{O}\left(\frac{n^8 n_\tau^2 m^2}{\lambda_{\min}^2 \epsilon^4 \delta}\right),$$

where $\lambda_{\min}$ is the smallest eigenvalue (or singular value) of the covariance matrix, $\Sigma_{\tau a}$, over all (τ, a) pair. The resulting sample complexity of exploration of KWIK-Rmax in this class of MDPs is

$$\tilde{\mathcal{O}}\left(\frac{n^8 n_\tau^2 m^2 {V_{\max}}^5}{\lambda_{\min}^4 \epsilon^5 (1-\gamma)^5 \delta}\right).$$

The large exponent in the KWIK bound above is due to the need for learning the covariance matrices, $\Sigma_{\tau a}$. If only the mean vector in the offset dynamics are unknown and the covariance matrix is assumed to be $\sigma_{\tau a} I_n$ for the type–action pair (τ, a), as assumed by Abbeel and Ng [2005], then applying input-partition to the KWIK bound in Equation 5.6 yields the following KWIK bound for learning this class of MDPs,

$$\mathcal{O}\left(\frac{nm\tau}{\epsilon^2\delta}\right),$$

and the sample complexity of exploration of KWIK-Rmax is

$$\tilde{\mathcal{O}}\left(\frac{nmn_\tau {V_{\max}}^3}{\epsilon^3(1-\gamma)^3\delta}\right).$$

7.2.4 Factored-State MDPs with Known Structure

In many applications, the MDP's state space is most naturally represented as the cross product of subspaces, each of which corresponds to a state variable. Under such conditions, dynamic programming and reinforcement learning often face the "curse of dimensionality" [Bellman 1957], which says that the number of states in the MDP is exponential in the number of state variables, rendering most finite MDP-based learning/optimization algorithms intractable. Factored-state representations [Boutilier et al. 1999] are compact representations of MDPs that avoid explicitly enumerating all states when defining an MDP. Although planning in factored-state MDPs remains hard in the worst case [Littman 1997], their compact structure can reduce the sample complexity of learning significantly since the number of parameters needed to specify a factored-state MDP can be exponentially smaller than in the unstructured case.

In a factored-state MDP, let $m = |\mathcal{A}|$ be the number of actions; every state is a vector consisting of n component: $s = (s[1], s[2], \ldots, s[n]) \in \mathcal{S}$. Each component $s[i]$ is called a *state variable* and can take values in a finite set $\mathcal{S}_i$. The whole state space $\mathcal{S}$ is thus $\mathcal{S}_1 \times \cdots \times \mathcal{S}_n$. Without loss of generality, assume $N = |\mathcal{S}_i|$ for all i and thus $|\mathcal{S}| = N^n$. The transition function is factored into the product of n transition functions, one for each state variable:

$$T(s' \mid s, a) = \prod_{i=1}^{n} T_i(s'[i] \mid s, a).$$

In other words, the values of all the next state variables are independent of each other, conditioned on the current state–action pair (s, a). Furthermore, we assume that the distribution $T_i(\cdot \mid s, a)$ depends on a small subset of $\{s[1], s[2], \ldots, s[n]\}$, denoted $\mathcal{P}(i)$. This assumption is valid in many real-life applications and and essential in most work in graphical models (*e.g.*, Abbeel et al. [2006a]). Let D be the largest size of $\mathcal{P}_i$: $D = \max_i |\mathcal{P}_i|$. Although we treat D as a constant, we include D explicitly in the KWIK bounds to show how D affects learning efficiency.

Using the quantities defined above, the transition function can be rewritten as

$$T(s' \mid s, a) = \prod_{i=1}^{n} T_i(s'[i] \mid \mathcal{P}(i), a).$$

An advantage of this succinct representation is that, instead of representing the complete conditional probability table, $T_i(\cdot \mid s, a)$, which has $N^n m$ entries, we only need to use the smaller $T_i(\cdot \mid \mathcal{P}(i), a)$, which has at most $N^D m$ entries. If $D \ll n$, we are able to achieve an exponentially more compact representation. These kinds of transition functions can be represented as *dynamic Bayesian networks* or *DBNs* [Dean and Kanazawa 1989]. Here, we consider the case where the structure (namely, $\mathcal{P}(i)$ for all i) is known *a priori* and show how to relax this assumption in §7.2.5.

The reward function can be represented in a similar way as the sum of *local reward functions*. We assume, for simplicity, that $R(s, a)$ is known and focus on KWIK-learning the transition functions. Our algorithm and insights still apply when the reward function is unknown.

Transitions in a factored-state MDP can be thought of as mappings from vectors $(s[1], s[2], \ldots, s[n], a)$ to vectors $(s'[1], s'[2], \ldots, s'[n])$. Given known dependencies, cross-product with $\alpha = 1/n$ can be used to learn each component of the transition function to guarantee that the combined transition distribution differs from the true transition distribution by ϵ in terms of ℓ_1 distribution:

Lemma 32 *Let P and Q be two probability distributions over the same finite sample*

space Ω^n, which are factored: for any $\omega = (\omega_1, \ldots, \omega_n) \in \Omega^n$,

$$\begin{aligned} P(\omega) &= \prod_{i=1}^{n} P_i(\omega_i), \\ Q(\omega) &= \prod_{i=1}^{n} Q_i(\omega_i), \end{aligned}$$

for some probability distributions P_i and Q_i over Ω. Then,

$$|P - Q| \leq \sum_{i=1}^{n} |P_i - Q_i| .$$

The proof is in §7.4.2. Each component transition probability function $T_i(\cdot \mid \cdot, \cdot)$ can be learned by input-partition applied to dice-learning. This three-level KWIK algorithm provides an approach to learn the transition function of a factored-state MDP with the following KWIK bound:

$$\begin{aligned} B_{\text{factored-state MDP}}(\epsilon, \delta) &= \sum_{i=1}^{n} B_{\text{input-partition}}\left(\frac{\epsilon}{n}, \frac{\delta}{n}\right) \\ &= nN^D m \cdot B_{\text{dice-learning}}\left(\frac{\epsilon}{n}, \frac{\delta}{nN^D m}\right) \\ &= \frac{n^3 m N^{D+1}}{\epsilon^2} \ln \frac{nN^{D+1}m}{\delta} \\ &= \mathcal{O}\left(\frac{n^3 m D N^{D+1}}{\epsilon^2} \ln \frac{nmN}{\delta}\right), \end{aligned}$$

where the first equality applies output-combination with $\alpha = 1/n$; the second equality applies input-partition to all possible (s, a) pairs, *ignoring non-parent variables*, and there are $N^D m$ many of them; the third equality simply uses the KWIK bound of dice-learning.

Combined with Theorem 21, the KWIK bound above results in the following sample complexity of exploration of KWIK-Rmax:

$$\mathcal{O}\left(\frac{n^3 m D N^{D+1} {V_{\max}}^3}{\epsilon^3 (1-\gamma)^3} \ln \frac{nmN}{\delta} \ln \frac{1}{\epsilon(1-\gamma)}\right) = \tilde{\mathcal{O}}\left(\frac{n^3 m D N^{D+1} {V_{\max}}^3}{\epsilon^3 (1-\gamma)^3}\right).$$

This insight can be used to derive the factored-state-MDP learning algorithm proposed by Kearns and Koller [1999].

7.2.5 Factored-State MDPs with Unknown Structures

Without known structural dependencies of the DBN, learning a factored-state MDP is more challenging. Strehl et al. [2007] showed that each possible dependence structure

can be viewed as a separate hypothesis and provided an algorithm for learning the dependencies in a factored-state MDP while learning the transition probabilities. The resulting KWIK bound is super-quadratic in $k = \Theta(2^D)$, where D, as before, is the maximum in-degree of the true DBN.

We can construct a conceptually simpler algorithm for this problem using components introduced throughout this paper. The key idea is to insert a noisy union component between cross-product and input-partition in the three-leval algorithm in §7.2.4. As a whole, the algorithm has four levels with a cross-product at the top to decompose the transitions for the separate components of the factored-state representation, as in §7.2.4. Each of these n individual factor transitions is learned using a separate copy of the noisy union algorithm. Within each of these copies, a union is performed over the $k = \binom{n}{D} = \mathcal{O}(n^D)$ possible parent sets for the given state component. As in the known-structure case, an input-partition algorithm is used for each of those possible configurations to handle the different combinations of parent values and action, and finally dice-learning is used to learn the individual transition probabilities themselves. The resulting KWIK bound is:

$$\begin{aligned}
B_{\text{factored-state MDP}} &= \sum_{i=1}^{n} B_{\mathsf{noisy-union}}\left(\frac{\epsilon}{n}, \frac{\delta}{n}\right) \\
&= \mathcal{O}\left(n\left(\frac{kn^2}{\epsilon^2}\ln\frac{kn}{\delta} + \sum_{i=1}^{k} B_{\mathsf{input-partition}}\left(\frac{\epsilon}{8n}, \frac{\delta}{n(k+1)}\right)\right)\right) \\
&= \mathcal{O}\left(\frac{kn^3}{\epsilon^2}\ln\frac{kn}{\delta} + nkN^D m \cdot B_{\mathsf{dice-learning}}\left(\frac{\epsilon}{8n}, \frac{\delta}{n(k+1)mN^D}\right)\right) \\
&= \mathcal{O}\left(\frac{kn^3}{\epsilon^2}\ln\frac{kn}{\delta} + nmkN^D \cdot \frac{Nn^2}{\epsilon^2}\ln\frac{nmkN^{D+1}}{\delta}\right) \\
&= \mathcal{O}\left(\frac{n^{D+3}mDN^{D+1}}{\epsilon^2}\ln\frac{nmN}{\delta}\right),
\end{aligned}$$

where the first equality applies output-combination with $\alpha = 1/n$; the second equality uses noisy-union; the third equality applies input-partition to all possible (s, a) pairs, *ignoring non-parent variables*, and there are $N^D m$ many of them; and the fourth equality simply uses the KWIK bound of dice-learning.

The resulting sample complexity of exploration of KWIK-Rmax is thus

$$\tilde{\mathcal{O}}\left(\frac{n^{D+3}mDN^{D+1}{V_{\max}}^2}{\epsilon^2(1-\gamma)^2}\right).$$

Note that our noisy union component is conceptually simpler, significantly more efficient ($k \ln k$ vs. $k^2 \ln k$ dependence on $k = \binom{n}{D}$), and more generally applicable than the similar procedure employed by Strehl et al. [2007]. Thus, our improvement is exponential in D and polynomial in n. Empirical studies in §7.3.1 show that the new algorithm works much better in practice.

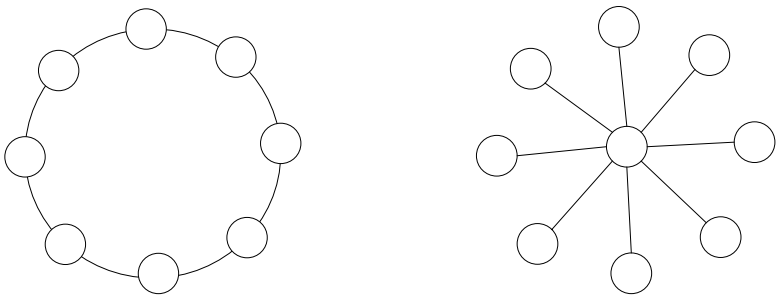
7.3 Case Studies

We have covered a number of special classes of MDPs that are KWIK-learnable and thus can apply KWIK-Rmax to explore efficiently in such MDPs. This section shows the resulting algorithms demonstrate significant performance improvements over previous methods in the literature, in addition to their theoretical contributions. Two representative examples are chosen: (ii) system administrator for factored-state MDP without known DBN structure, and (ii) robotics navigation with unknown, typed, normal offset; detailed descriptions and more results for these two problems are found in Diuk et al. [2009] and Brunskill et al. [2008], respectively. Interested readers are referred to more empirical studies in other papers [Walsh et al. 2009a; Leffler et al. 2007].

7.3.1 System Administrator

In the system-administrator problem [Guestrin et al. 2003], an agent (the system administrator) attempts to maintain a network of n computers, each of which is connected to some other computers. For instance, the machines may be connected as a ring or a star (Figure 7.1). The working status of a computer is encoded as a binary number: 0 for failed and 1 for working. The reward received by the agent at any timestep is proportional to the number of working machines. If a machine fails, the agent may reboot it so that its status is very likely to be working in the next timestep. If a machine is not rebooted, its status in the next timestep depends stochastically on the status of its parents (namely, the machines that are connected to it). In every timestep, the agent can reboot at most one machine.

This problem can be formulated as an MDP as follows. The state space $\mathcal{S} = \mathbb{B}^n$



(a) "Ring" topology (b) "Star" topology

Figure 7.1: Example network topologies in the system administrator problem.

contains all possible joint statuses of the set of n machines. The action set $\mathcal{A} = \{0, 1, \cdots, n\}$ contains $n+1$ actions, including the option of rebooting no machines and the options of rebooting any one of the n machines. The reward function is

$$R(s, a) \stackrel{\text{def}}{=} K_r \sum_{i=1}^{n} s[i],$$

for some constant $K_r > 0$. The transition function is factored:

$$T(s' \mid s, a) \stackrel{\text{def}}{=} \prod_{i=1}^{n} T_i(s'[i] \mid \mathcal{P}(i), a),$$

where $\mathcal{P}(i)$ is the set of computers connected to machine i, and $T_i(s'[i] \mid \mathcal{P}(i), a)$ specifies the "local" transition probabilities of machine i:

- Rebooting machine i at timestep t always guarantees its status to be 1 (working) at timestep $t+1$;
- If machine i is down at timestep t and is not rebooted, it remains down at timestep $t+1$;
- If machine i is working at timestep t, the probability of its status being 0 at timestep $t+1$ is $\min\{1, K_0 + K_1 \sum_{j \in \mathcal{P}(i)} (1 - s[j])\}$ for some constants $K_0, K_1 > 0$. That is, if all neighboring machines are working, there is a positive probability K_0 for machine i to fail. With more and more neighboring machines having failed, the probability of failing machine i also increases. There are of course other ways to define transition probabilities.

The ring network (Figure 7.1) with $n = 8$ was chosen for experiments. Therefore, the in-degree D of the DBN of this MDP was always 3, including two neighbors and

the machine itself. Parameters were set by: $K_r = 1$, $K_0 = 0.05$, and $K_1 = 0.3$. We compared three algorithms:

- Factored-Rmax [Strehl 2007a] is the three-layer instantiation of KWIK-Rmax described in §7.2.4. The structure of the DBN is given to the algorithm as input, so there is no need for structure discovery.
- SLF-Rmax [Strehl et al. 2007] is similar to Factored-Rmax but does not require a known DBN structure. It uses a subroutine to discovery the structure of the DBN.
- Met-Rmax [Diuk et al. 2009] is the four-layer instantiation of KWIK-Rmax described in §7.2.5. It is similar to SLF-Rmax but uses a different approach (*i.e.*, noisy union) to discover the DBN structure.

All three algorithms require the "knownness parameter" m, and SLF-Rmax requires an additional parameter ϵ_1 in structure discovery. A parameter search was performed for the three algorithms to optimize them: $m = 30$ for factored-Rmax, $m = 30$ and $\epsilon_1 = 0.2$ for SLF-Rmax, and $m = 50$ for met-Rmax.

Figure 7.2 shows the results. As expected, Factored-Rmax was the fastest as it had access to the information of DBN structure, which was unavailable to the other two algorithms. Met-Rmax is able to discover the underlying DBN structure like SLF-Rmax but at a much faster rate, which is consistent with our analysis in §7.2.5.

7.3.2 Robotics Navigation

The second experiment is in a real-life robotic environment involving a navigation task where a robotic car must traverse multiple surface types to reach a goal location. This experiment is to demonstrate the noisy offset dynamics described in §7.2.3 can provide a sufficiently good representation of real-world dynamics to allow a robot to learn good navigation policies.

Previous study has demonstrated the benefits of using types of states [Leffler et al. 2005; 2007]. Specifically, Leffler et al. [2007] proposes the RAM-Rmax algorithm—a variant of Rmax that is able to make use of state types and enjoys a smaller sample complexity that depends on the number of types, which can be much smaller than the

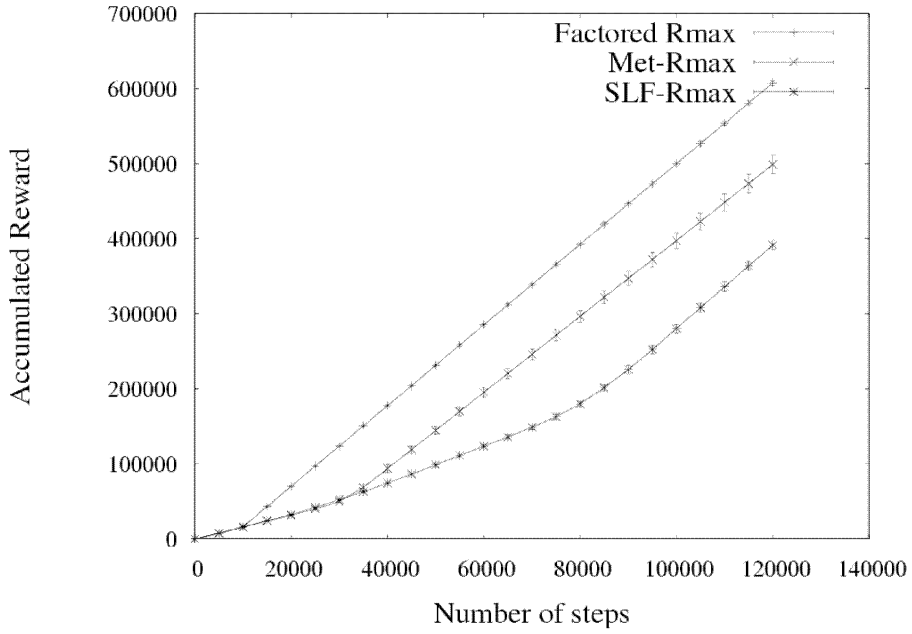


Figure 7.2: Cumulative reward on each timestep for the three algorithms on the SysAdmin domain: factored-Rmax (with given DBN structure), met-Rmax and SLF-Rmax.

number of states. However, RAM-Rmax is for finite MDPs. Applying it in continuous MDPs requires a pre-processing step of discretization. Here, we assume the state offset in this robotics navigation task follows the typed normal distributions defined in §7.2.3 (a common and reasonable assumption in many robotics applications), and compared an algorithm called CORL [Brunskill et al. 2008] to RAM-Rmax. CORL stands for "Continuous-state Offset-dynamics Reinforcement Learner", and is an instantiation of KWIK-Rmax for the class of MDPs with typed, normal offsets. It works directly with continuous-valued states, thus removing the need for discretization.

A LEGO® Mindstorms NXT robot (Figure 7.3(a)) was run on a maze-like environment with two types of surface (Figure 7.3(b)): rocks embedded in wax and a carpeted area. The task of the robot was to navigate in this environment from the start state to the goal state without going outside the environment's boundaries. A tracking pattern was placed on the top of the robot and an overhead camera was used to determine the robot's current position and orientation. The tracking pattern was used to avoid the complications of robot localization (see, *e.g.*, Thrun et al. [2001]), and to allow us to focus on how well the algorithms explores in this (approximately) Markovian problem.

(a) A LEGO® robot.

(b) Image of the environment: the start location and orientation is marked with an arrow, and the goal location is indicated by the circle.

Figure 7.3: Robot in the navigation experiment.

The state space has three components: the x-coordinate, the y-coordinate, and the orientation θ. The reward function assigns -1 for going out of boundaries, $+1$ for reaching the goal, and -0.01 for taking an action. The reward function was known to both algorithms, and so the only target to learn was the transition dynamics. Reaching the goal and going out of boundaries would terminate the current episode and the robot was moved back to the start location and orientation for a new episode. Three actions were allowed: *going forward*, *turning left*, and *turning right*. The discount factor $\gamma = 1$ was used since this task is episodic.

In our experiments, RAM-Rmax used value iteration over the finite, discretized state space, while CORL used fitted value iteration (§3.3.2) whose prototypical points are the same as the discretized states used by RAM-Rmax. Both algorithms used an EDISON image segmentation system to uniquely identify the current surface type. Parameters needed by both algorithms were hand-tuned.

Figure 7.4(a) shows the average reward with standard deviation for each of the algorithms over three runs. Both algorithms are able to receive near-optimal reward on a consistent basis, choosing similar paths to the goal. This result indicates our dynamics representation is sufficient to allow our algorithm to learn well in this real-life environment. Combined with previous study by Leffler et al. [2007], it suggests that:

- Rmax-style exploration not only possesses strong theoretical guarantees, but can

also find near-optimal policies faster than popular exploration heuristics in practice; and

- Rmax-style exploration can be further improved by the use of generalization, either via types or via making structural assumptions about the MDP's dynamics.

In addition, a compact parametric representation may also lead to computational benefits, as shown in Figure 7.4(b). The computational time per episode was roughly constant for CORL, but grew with the number of episodes in RAM-Rmax (as a result of its dynamics model representation).

Finally, we note that the type function in our experiments was assumed to be known, which may not always be true in other applications. A preliminary study on type discovery was recently done by Diuk et al. [2009] using noisy union in §6.5.

7.4 Proofs

This section provides detailed proofs of all technical lemmas used in this chapter.

7.4.1 Proof of Theorem 21

The proof relies on a general PAC-MDP result in Theorem 4, in which the set K_t of state–action pairs is defined as follows.

Definition 14 *Let $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ be an MDP. At timestep t of KWIK-Rmax (Algorithm 25), define the set of* known state–actions with respect to $\mathbf{A}_T$ and $\mathbf{A}_R$ *by*

$$K_{\mathbf{A}_T, \mathbf{A}_R} \stackrel{\text{def}}{=} \{(s,a) \in \mathcal{S} \times \mathcal{A} \mid \mathbf{A}_T(s,a) \neq \perp, \mathbf{A}_R(s,a) \neq \perp\}.$$

In other words, $K_{\mathbf{A}_T, \mathbf{A}_R}$ consists of state–action pairs where both KWIK algorithms, $\mathbf{A}_T$ and $\mathbf{A}_R$, are able to make valid predictions.

We first provide a version of the simulation lemma (Lemma 33), and then verify the three conditions in Theorem 4 in Lemmas 34–36 to show KWIK-Rmax is PAC-MDP. We choose the five parameters by

$$\epsilon_R = \frac{\epsilon(1-\gamma)}{20}, \quad \epsilon_T = \frac{\epsilon(1-\gamma)}{20\gamma V_{\max}}, \quad \epsilon_P = \frac{\epsilon(1-\gamma)}{20}, \quad \delta_T = \delta_R = \frac{\delta}{4}. \tag{7.2}$$

Lemma 33 *(Simulation Lemma) Let* $M_1 = \langle \mathcal{S}, \mathcal{A}, T_1, R_1, \gamma \rangle$ *and* $M_2 = \langle \mathcal{S}, \mathcal{A}, T_2, R_2, \gamma \rangle$ *be two MDPs with the same state/action spaces and discount factor. Let* Q_1^* *and* Q_2^* *(*V_1^* *and* V_2^**) be their optimal state–action value (state-value) functions, respectively. Assume the two transition functions and reward functions are close in the following sense: there exist two constants,* ϵ_T *and* ϵ_R*, such that for every* (s, a)*,*

$$\begin{aligned} |T_1(\cdot \mid s, a) - T_2(\cdot \mid s, a)| &\leq \epsilon_T \\ |R_1(s, a) - R_2(s, a)| &\leq \epsilon_R, \end{aligned}$$

where $|T_1(\cdot \mid s, a) - T_2(\cdot \mid s, a)|$ *is defined in Definition 13. Then, for any* $s \in \mathcal{S}$ *and* $a \in \mathcal{A}$*:*

$$\begin{aligned} |Q_1^*(s, a) - Q_2^*(s, a)| &\leq \frac{\epsilon_R + \gamma V_{\max} \epsilon_T}{1 - \gamma} \\ |V_1^*(s) - V_2^*(s)| &\leq \frac{\epsilon_R + \gamma V_{\max} \epsilon_T}{1 - \gamma}. \end{aligned}$$

Proof. We will prove the case where $\mathcal{S}$ is uncountable; the other case is similar. Define the *Bellman operators*, $\mathfrak{B}_1$ and $\mathfrak{B}_2$, for M_1 and M_2, respectively: for $i = 1, 2$ and any state–action value function $Q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$,

$$\mathfrak{B}_i Q(s, a) \stackrel{\text{def}}{=} R_i(s, a) + \gamma \int_{s' \in \mathcal{S}} T_i(s' \mid s, a) \sup_{a' \in \mathcal{A}} Q(s', a') ds'.$$

It is known that Q_i^* is the *fixed point* of $\mathfrak{B}_i$: $\mathfrak{B}_i Q_i^* = Q_i^*$. Define two errors: the ℓ_∞ approximation error $e = \|Q_1^* - Q_2^*\|_\infty$ and the ℓ_∞ Bellman backup error $b = \|\mathfrak{B}_1 Q_2^* - \mathfrak{B}_2 Q_2^*\|_\infty$. Then,

$$\begin{aligned} e &= \|\mathfrak{B}_1 Q_1^* - \mathfrak{B}_2 Q_2^*\|_\infty \\ &\leq \|\mathfrak{B}_1 Q_1^* - \mathfrak{B}_1 Q_2^*\|_\infty + \|\mathfrak{B}_1 Q_2^* - \mathfrak{B}_2 Q_2^*\|_\infty \\ &\leq \gamma \|Q_1^* - Q_2^*\|_\infty + \|\mathfrak{B}_1 Q_2^* - \mathfrak{B}_2 Q_2^*\|_\infty \\ &= \gamma e + b, \end{aligned}$$

where the first step is due to the fixed-point property of $\mathfrak{B}_i$, the second due to the triangle inequality, the third due to the contraction property of $\mathfrak{B}_i$ (Lemma 5), and the last due to the definitions of e and b. It follows immediately that $(1 - \gamma)e \leq b$, and so

$$e \leq \frac{b}{1 - \gamma}. \tag{7.3}$$

We now give an upper bound for b:

$$\begin{aligned}
b &= \sup_{s,a} |\mathfrak{B}_1 Q_2^*(s,a) - \mathfrak{B}_2 Q_2^*(s,a)| \\
&= \sup_{s,a} \left| (R_1(s,a) - R_2(s,a)) + \gamma \int_{\mathcal{S}} \big(T_1(s' \mid s,a) - T_2(s' \mid s,a)\big) \sup_{a'} Q_2^*(s',a') ds' \right| \\
&\leq \sup_{s,a} |R_1(s,a) - R_2(s,a)| + \gamma \sup_{s,a} \left| \int_{\mathcal{S}} \big(T_1(s' \mid s,a) - T_2(s' \mid s,a)\big) \sup_{a'} Q_2^*(s',a') ds' \right| \\
&\leq \epsilon_R + \gamma \sup_{s,a} \int_{\mathcal{S}} \big|T_1(s' \mid s,a) - T_2(s' \mid s,a)\big| \sup_{a'} \big|Q_2^*(s',a')\big| \, ds' \\
&\leq \epsilon_R + \gamma V_{\max} \sup_{s,a} \int_{\mathcal{S}} \big|T_1(s' \mid s,a) - T_2(s' \mid s,a)\big| \, ds' \\
&\leq \epsilon_R + \gamma V_{\max} \epsilon_T,
\end{aligned}$$

where the first inequality is due to the triangle inequality and the fact that $\sup_x \{f_1(x) + f_2(x)\} \leq \sup_x f_1(x) + \sup_x f_2(x)$ for all real-valued functions f_1 and f_2, the second due to the Cauchy-Schwartz inequality, the third due to $\|Q_2^*\|_\infty \leq V_{\max}$ by assumption (§2.2). Combining this result with Equation 7.3, we have for all (s,a) that

$$|Q_1^*(s,a) - Q_2^*(s,a)| \leq e \leq \frac{b}{1-\gamma} \leq \frac{\epsilon_R + \gamma V_{\max} \epsilon_T}{1-\gamma}.$$

The second part of the lemma follows immediately from the following relation between optimal state–action value functions and optimal state-value functions: for any $s \in \mathcal{S}$,

$$|V_1^*(s) - V_2^*(s)| = \left| \sup_a Q_1^*(s,a) - \sup_a Q_2^*(s,a) \right| \leq \sup_a |Q_1^*(s,a) - Q_2^*(s,a)| .$$

□

Lemma 34 *With probability at least $1 - \delta/2$, $Q_t(s,a) \geq Q^*(s,a) - \epsilon/4$ for all t and (s,a).*

Proof. Since the algorithm computes an ϵ_P-accurate Q_t function, we have

$$Q_t(s,a) - Q^*(s,a) \geq Q_{\hat{M}}^*(s,a) - \epsilon_P - Q^*(s,a).$$

We next bound $Q_{\hat{M}}^*(s,a) - Q^*(s,a)$. Let M_{K_t} be the known state–action MDP, where K_t is given in Definition 14. The transition and reward functions in M_{K_t} agree with $\hat{M}$ in unknown state–actions and with M in known state–actions. Since the transition and

reward functions of $\hat{M}$ are accurate (with failure probability at most $\delta_R + \delta_T = \delta/2$), Lemma 33 yields

$$\left|Q^*_{\hat{M}}(s,a) - Q^*_{M_K}(s,a)\right| \leq \frac{\epsilon_R + \gamma V_{\max}\epsilon_T}{1-\gamma}.$$

On the other hand, since M_{K_t} is identical to M except in unknown state–actions to which the largest possible values are assigned, its optimal state–action value function must be optimistic: $Q^*_{M_K}(s,a) \geq Q^*(s,a)$ for all (s,a). Combining these inequalities, we have

$$\begin{aligned} Q_t(s,a) - Q^*(s,a) &\geq Q^*_{\hat{M}}(s,a) - \epsilon_P - Q^*(s,a) \\ &\geq Q^*_{M_K}(s,a) - \frac{\epsilon_R + \gamma V_{\max}\epsilon_T}{1-\gamma} - \epsilon_P - Q^*(s,a) \\ &\geq -\frac{\epsilon_R + \gamma V_{\max}\epsilon_T}{1-\gamma} - \epsilon_P. \end{aligned}$$

The lemma follows by the parameters given in Equation 7.2. □

Lemma 35 *With probability at least $1 - \delta/2$, $V_t(s_t) - V^{\pi_t}_{M_K}(s_t) \leq \epsilon/4$, where $V_t(s) = \max_a Q_t(s,a)$, and π_t is the policy computed by* KWIK-Rmax *at timestep t.*

Proof. If Q_t is ϵ_P-accurate, then V_t is also ϵ_P-accurate:

$$\left|V_t(s) - V^*_{\hat{M}}(s)\right| = \left|\sup_a Q_t(s,a) - \sup_a Q^*_{\hat{M}}(s,a)\right| \leq \sup_a \left|Q_t(s,a) - Q^*_{\hat{M}}(s,a)\right| \leq \epsilon_P.$$

Consequently,

$$\begin{aligned} V_t(s) - V^{\pi_t}_{M_K}(s) \;&\leq\; V_t(s) - V^{\pi_t}_{\hat{M}}(s) + \frac{\epsilon_R + \gamma V_{\max}\epsilon_T}{1-\gamma} \\ &\leq V^*_{\hat{M}}(s) + \epsilon_P - V^{\pi_t}_{\hat{M}}(s) + \frac{\epsilon_R + \gamma V_{\max}\epsilon_T}{1-\gamma} \\ &\leq \frac{2\epsilon_P}{1-\gamma} + \epsilon_P + \frac{\epsilon_R + \gamma V_{\max}\epsilon_T}{1-\gamma} \\ &\leq \frac{\epsilon}{4}, \end{aligned}$$

where the first step is from Lemma 33, and the second from the ϵ_P-accuracy of V_t in MDP $\hat{M}$, the third is due to an inequality of Singh and Yee [1994],[2], and the last is

[2]Although Singh and Yee [1994] consider finite MDPs only, their proof is also valid for arbitrary MDPs as long as the value functions and state occupation distributions used in the proof are all well-defined in a continuous-state MDP.

due to Equation 7.2. The only failure probability is in the use of Lemma 33, where the KWIK learners, $\mathbf{A}_T$ or $\mathbf{A}_R$, may fail to return accurate prediction in some state–actions; this failure probability is at most $\delta_T + \delta_R = \delta/2$, by a union bound. □

Lemma 36 *The total number of timesteps in which Q_t changes or an unknown state is visited, denoted by $\zeta(\epsilon, \delta)$, is at most $B_T(\epsilon_T, \delta_T) + B_R(\epsilon_R, \delta_R)$.*

PROOF. Since Q_t (and also $\hat{M}$) is unchanged unless K_t changes (as a result of experiencing some unknown (s_t, a_t)), we may only bound the number of timesteps in which an unknown state–action pair is experienced, which is $B_T(\epsilon_T, \delta_T) + B_R(\epsilon_R, \delta_R)$. □

We can now complete the proof of Theorem 21 by the previous lemmas and Theorem 4, yielding the desired sample complexity of exploration in Equation 7.1.

7.4.2 Proof of Lemma 32

PROOF (of Lemma 32). For $j = 0, 1, 2, \ldots, n$, define the following factored distribution over Ω^n: for any $\omega = (\omega_1, \ldots, \omega_n) \in \Omega^n$,

$$Q^{(j)}(\omega) = \prod_{i=1}^{j} Q_i(\omega_i) \prod_{i=j+1}^{n} P_i(\omega_i).$$

Therefore, $Q^{(0)} = P$, and $Q^{(n)} = Q$. Furthermore, $Q^{(j-1)}$ and $Q^{(j)}$ only differ in the transition in the j-th factor. For any $j = 0, 1, \ldots, n-1$, we have

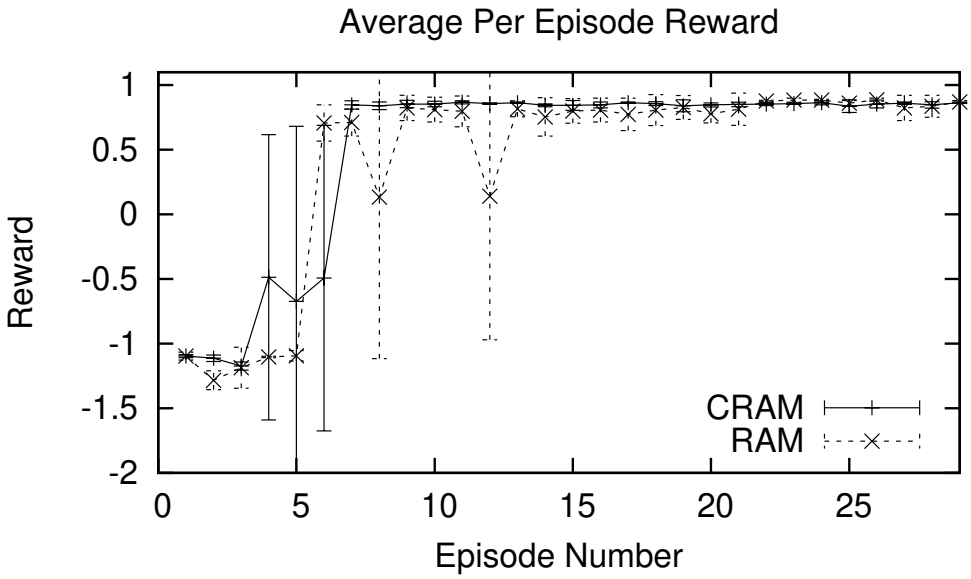
$$\begin{aligned}
\left|Q^{(j)} - Q^{(j+1)}\right| &= \sum_{\omega_1,\ldots,\omega_n \in \Omega} \left|Q^{(j)}((\omega_1, \ldots, \omega_n)) - Q^{(j+1)}((\omega_1, \ldots, \omega_n))\right| \\
&= \sum_{\omega_1,\ldots,\omega_n \in \Omega} Q_1(\omega_1) \cdots Q_j(\omega_j) P_{j+2}(\omega_{j+2}) \cdots P_n(\omega_n) \left|P_{j+1}(\omega_{j+1}) - Q_{j+1}(\omega_{j+1})\right| \\
&= \sum_{\omega_1 \in \Omega} Q_1(\omega_1) \cdots \sum_{\omega_j \in \Omega} Q_j(\omega_j) \sum_{\omega_{j+2} \in \Omega} P_{j+2}(\omega_{j+2}) \cdots \sum_{\omega_n \in \Omega} P_n(\omega_n) \\
&\quad \cdot \sum_{\omega_{j+1} \in \Omega} \left|P_{j+1}(\omega_{j+1}) - Q_{j+1}(\omega_{j+1})\right| \\
&= \sum_{\omega_{j+1} \in \Omega} \left|P_{j+1}(\omega_{j+1}) - Q_{j+1}(\omega_{j+1})\right| \\
&= \left|P_{j+1} - Q_{j+1}\right|,
\end{aligned}$$

where the first equality is by the definition of the ℓ_1-distance, the second is by the definition of $Q^{(j)}$, and the last follows from the fact that P_i and Q_i defines a marginal distribution over variable ω_i.

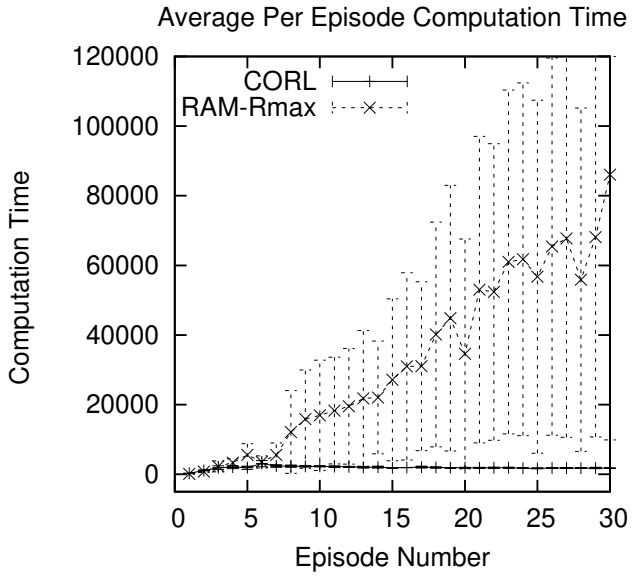
Applying this equation for $j = 0, 1, \ldots, n-1$ and the triangle inequality, we have

$$\begin{aligned} |P - Q| &= \left|Q^{(0)} - Q^{(n)}\right| \\ &= \left|Q^{(0)} - Q^{(1)} + Q^{(1)} - \cdots - Q^{(n-1)} + Q^{(n-1)} - Q^{(n)}\right| \\ &\leq \left|Q^{(0)} - Q^{(1)}\right| + \left|Q^{(1)} - Q^{(2)}\right| + \cdots + \left|Q^{(n-1)} - Q^{(n)}\right| \\ &= |P_1 - Q_1| + |P_2 - Q_2| + \cdots + |P_n - Q_n| \,. \end{aligned}$$

and finish the proof of the lemma. □



(a) Reward received by algorithms averaged over three runs. Error bars show one standard deviation.



(b) Total time taken by algorithms averaged over three runs.

Figure 7.4: Experimental results for the robot navigation problem. Error bars show one standard deviation.

Chapter 8

Model-free Approaches

This chapter studies model-free PAC-MDP algorithms. Although it is difficult to give a generic algorithmic scheme (like KWIK-Rmax in §7) for model-free algorithms due to technical difficulties in their analysis, we devise PAC-MDP model-free algorithms for certain types of MDPs and show the asymptotic optimality of one of them through a new lower bound for the sample complexity of exploration.

Most of the algorithms investigated in this chapter can be viewed as using KWIK techniques to approximate Bellman backups in a robust way to allow a polynomial sample complexity of exploration, including algorithms for finite MDPs in §§8.1–8.2 and an algorithm for finite-horizon MDPs with linear value-function approximation in §8.4. In §8.3, we also study an algorithm that is motivated by Rmax and generalizes it to the case where approximate policy iteration with linear value-function approximation is used for planning.

8.1 Approximate Real-Time Dynamic Programming

When an explicit description of an MDP model is available, asynchronous value iteration methods such as real-time dynamic programming can be used to compute the optimal value function (§3.1.2). In this and the next sections, we consider approximate real-time dynamic programming algorithms for finite MDPs, in which an approximate Bellman operator is repeatedly applied to refine the agent's state–action value function estimate in state–actions it visits online. Algorithms in these two sections can be shown to be PAC-MDP in a similar way based on the general PAC-MDP result (Theorem 4) and a key lemma given in §8.1.1.

In this section, we will assume that the agent is given an approximate MDP model

(§8.1.2) or a generative model (§8.1.3), with which approximate Bellman backups can be performed. Although these assumptions are not satisfied in many reinforcement-learning problems, these algorithms' analyses are instructive and can highlight the key components of the more complicated PAC-MDP analysis for the online reinforcement-learning algorithm studied in the next section. Furthermore, the PAC-MDP results in this section yield useful online performance guarantees for such *learning real-time heuristic search algorithms* that include real-time learning A* [Korf 1990] as a special case. Interested readers are referred to two variants of adaptive RTDP [Strehl et al. 2006a] that can be analyzed in a similar manner.

Unfortunately, approximate asynchronous value iteration is much harder to analyze than the model-based approaches considered in §7. In fact, its convergence is guaranteed only for finite MDPs and a few restricted cases when function approximation is used [Bertsekas and Tsitsiklis 1996]. We therefore focus on finite MDPs in this and the next sections. However, we note that generalization from finite MDPs to MDPs with state abstraction is possible, as is discussed in more detail in §8.2.3.

8.1.1 General Analysis

A common line of reasoning is adopted for the PAC-MDP analysis of the approximate real-time dynamic programming algorithms considered here and in the next section. These algorithms start with an optimistic state–action value function,

$$\forall (s, a) : Q_1(s, a) \leftarrow V_{\max},$$

and repeatedly apply an approximate Bellman operator to refine the Q-function in visited state–action pairs. Our analysis consists of three major steps that correspond to the three conditions in Theorem 4:

1. If the approximate Bellman backup is sufficiently accurate, say, using KWIK techniques described in §5 to estimate the Bellman backup values, we can guarantee that the value function remains optimistic after a Bellman backup, and thus satisfy the optimism condition of Theorem 4. For instance, the randomized RTDP and delayed Q-learning algorithms in §8.1.3 and §8.2 use KWIK algorithms and

analyses in §5.3.1 and §5.3.4, respectively.

2. The lemma given in this subsection is used to guarantee the accuracy condition of Theorem 4, and plays a role analogous to the simulation lemma for model-based algorithms. It makes use of a new notion of known state–actions based on Bellman errors, rather than on model prediction errors as in §7. This result is very general and can be applied to MDPs with infinite states.
3. Finally, we verify the bounded-surprises condition by bounding two terms: (i) the number of changes to the value function during a whole run of the algorithm, which relies on a "large-update" trick, and (ii) the number of visits to unknown states, which relies on the KWIK bound of estimating a Bellman backup value.

The second step is our focus here, while the other two are algorithm-specific and thus are left to their respective sections.

In model-based algorithms such as KWIK-Rmax, we define known state–actions to be those where accurate predictions about the transition and reward functions can be made with high confidence. The simulation lemma is then used to show that, if an agent follows a near-optimal policy in this known state–action MDP and has a small probability of reaching unknown state–actions, then the policy must be near-optimal in the true MDP.

In model-free algorithms such as Q-learning, a different notion of known state–actions is needed, because these algorithms do not explicitly estimate transition and reward functions of the MDP. Usually, model-free algorithms maintain an estimate of the optimal state–action value function, whose Bellman errors can be computed by Equation 2.14 and used to define known state–actions. Recall that the Bellman error of a value function in a state–action pair measures local consistency of this function. Below, known state–actions are defined to be those whose Bellman errors are essentially non-negative.

Definition 15 *Let Q be a state–action value function of an MDP $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, $\mathfrak{B}$ the Bellman operator defined by Equation 2.10, and $\varepsilon \in \mathbb{R}_+$ a parameter, the set of*

known state–actions with respect to Q and ε *is defined by*

$$K_Q(\varepsilon) \stackrel{\text{def}}{=} \{(s,a) \in \mathcal{S} \times \mathcal{A} \mid \mathfrak{B}Q(s,a) - Q(s,a) \geq -\varepsilon\}.$$

With Definition 15, an important lemma can be stated and proved that is useful for analyzing model-free algorithms.

Lemma 37 *Using the notation in Theorem 4 and Definition 15, we define* $K_t \stackrel{\text{def}}{=} K_{Q_t}(\epsilon(1-\gamma)/4)$. *Then, the accuracy condition in Theorem 4 is satisfied.*

Proof. For convenience, denote $\epsilon_1 = \epsilon(1-\gamma)/4$. Due to the construction of the known state–action MDP, we have

$$\mathfrak{B}_{M_K,\pi_t} Q_t(s,a) - Q_t(s,a) \begin{cases} \geq -\epsilon_1, & \text{if } (s,a) \in K_t \\ 0, & \text{otherwise} \end{cases}$$

where $\mathfrak{B}_{M_K,\pi_t}$ is the Bellman operator defined using M_K and policy π_t:

$$\mathfrak{B}_{M_K,\pi_t} Q_t(s,a) \stackrel{\text{def}}{=} \begin{cases} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \left(T(s' \mid s,a) Q_t(s', \pi_t(s'))\right), & \text{if } (s,a) \in K_t \\ Q_t(s,a)(1-\gamma) + \gamma Q_t(s,a), & \text{otherwise.} \end{cases}$$

Therefore, $Q_t \leq \mathfrak{B}_{M_K,\pi_t} Q_t + \epsilon_1$. Due to the two properties of the Bellman operator given in Lemmas 4 and 6, applying the operator to the inequality above repeatedly yields, for every (s,a), that

$$\begin{aligned} Q_t(s,a) &\leq \mathfrak{B}^2_{M_K,\pi_t} Q_t(s,a) + \gamma\epsilon_1 + \epsilon_1 \\ &\leq \mathfrak{B}^3_{M_K,\pi_t} Q_t(s,a) + \gamma^2\epsilon_1 + \gamma\epsilon_1 + \epsilon_1 \\ &\leq \cdots \\ &\leq \mathfrak{B}^\infty_{M_K,\pi_t} Q_t(s,a) + \epsilon_1(1 + \gamma + \gamma^2 + \cdots) \\ &\leq Q^{\pi_t}_{M_K}(s,a) + \frac{\epsilon_1}{1-\gamma} \\ &= Q^{\pi_t}_{M_K}(s,a) + \frac{\epsilon}{4}. \end{aligned}$$

Hence,

$$V_t(s_t) = \max_{a \in \mathcal{A}} Q_t(s_t,a) = Q_t(s_t,a_t) \leq Q^{\pi_t}_{M_K}(s_t,a_t) + \frac{\epsilon}{4} = V^{\pi_t}_{M_K}(s_t) + \frac{\epsilon}{4},$$

where we have used the fact that $a_t = \pi_t(s_t) = \operatorname{argmax}_{a \in \mathcal{A}} Q_t(s_t,a)$. □

8.1.2 Modified RTDP

As a simple application of Theorem 4 and Lemma 37, this section provides online performance guarantees of a variant of RTDP that only has access to an approximate MDP model, instead of an exact model. Our analysis results in useful theoretical insights of such learning real-time algorithms within the PAC-MDP framework. In some sense, our results provide polynomial mistake-bound results for such learning real-time heuristic search algorithms, where a "mistake" occurs when the algorithm's policy is not ϵ-optimal (*c.f.*, §4.4.4).

Algorithm 26 Modified RTDP.

```
0: Inputs: S, A, γ, B̂, ε1 ∈ R+.
1: Initialize Q(s, a) ← V_max for all (s, a) ∈ S × A.
2: Initialize the first state s1 ∈ S.
3: for t = 1, 2, 3, ... do
4:    Choose the greedy action a_t ← argmax_{a∈A} Q(s_t, a).
5:    Compute an attempted backup value: q ← B̂Q(s_t, a_t).
6:    if q ≤ Q(s_t, a_t) − ε1 then
7:       Q(s_t, a_t) ← q.
8:    end if
9:    Observe the next state s_{t+1} ~ T(· | s_t, a_t).
10: end for
```

Algorithm 26 gives the modified RTDP algorithm, which requires as input an oracle $\hat{\mathfrak{B}}$ for performing approximate Bellman backups. The algorithm differs from RTDP in that it only updates a state–action's Q-value when the value decreases at least by ϵ_1 after the approximate Bellman backup, as required in Line 6; in other words, only "large" updates are allowed. Since rewards are non-negative, the value function must be non-negative as well. The modification to disallow small updates in the value function guarantees that the state–action value function can be updated a finite number of times, a condition required by Theorem 4. This fact permits us to show modified RTDP's polynomial sample complexity of exploration.

Theorem 22 *Assume* $\epsilon_1 = \epsilon(1-\gamma)/8$ *and* $\left\|\hat{\mathfrak{B}}Q - \mathfrak{B}Q\right\|_\infty \leq \epsilon_1$ *for all* $Q(\cdot,\cdot)$ *such that* $0 \leq Q(s,a) \leq V_{\max}$*, then modified RTDP is PAC-MDP with the following sample*

complexity of exploration:

$$\mathcal{O}\left(\frac{V_{\max}}{\epsilon(1-\gamma)}\left(\frac{|\mathcal{S}|\,|\mathcal{A}|\,V_{\max}}{\epsilon(1-\gamma)}+\ln\frac{1}{\delta}\right)\ln\frac{1}{\epsilon(1-\gamma)}\right).$$

The proof relies on Theorem 4 by verifying that three conditions hold. The optimism condition is verified by the following lemma, whose proof is given in the appendix.

Lemma 38 *In modified RTDP, we have at all timesteps t that*

$$Q_t(s,a) \geq Q^*(s,a) - \frac{\epsilon_1}{1-\gamma} = Q^*(s,a) - \frac{\epsilon}{8}.$$

Proof (of Theorem 22). The optimism and accuracy conditions of Theorem 4 are verified by Lemma 38 and Lemma 37, respectively. For Condition 3, first observe that every visit to a state–action outside K_t results in an update of the state–action value. To see why, observe that the attempted Bellman backup value, q, computed in Line 5 of Algorithm 26 is small enough to trigger an update:

$$q - Q(s_t,a_t) = \hat{\mathfrak{B}}Q(s_t,a_t) - Q(s_t,a_t) \leq \mathfrak{B}Q(s_t,a_t) - Q(s_t,a_t) + \epsilon_1 \leq -\frac{\epsilon(1-\gamma)}{4} + \epsilon_1 = -\epsilon_1.$$

Therefore, the number of surprises coincides with the number of updates to the value function. Now, observe that each update of the state–action value results in a reduction of at least ϵ_1 and that the initial state–action value is at most $V_{\max}$. So, the number of updates in each state–action value is at most $\kappa \stackrel{\text{def}}{=} V_{\max}/\epsilon_1$, and the total number of updates in all state–action values is at most $|\mathcal{S}|\,|\mathcal{A}|\,\kappa$. Hence, the bounded-surprises condition of Theorem 4 is satisfied with

$$\zeta(\epsilon,\delta) = |\mathcal{S}|\,|\mathcal{A}|\,\kappa = \frac{|\mathcal{S}|\,|\mathcal{A}|\,V_{\max}}{\epsilon_1} = \frac{8\,|\mathcal{S}|\,|\mathcal{A}|\,V_{\max}}{\epsilon(1-\gamma)}.$$

□

8.1.3 Randomized RTDP

In computing a Bellman backup as in RTDP, the per-timestep sample complexity is linear in the number of states in the worst case. In randomized RTDP, the complexity is weakly dependent on the number of states when a generative model $\mathfrak{O}$ is available. This improvement is achieved by replacing the exact Bellman operator by a randomized

approximate Bellman operator, which draws m samples from a generative model and then averages the m sample backup values. Clearly, such a Monte Carlo approximation gives an unbiased estimate of the true Bellman backup, and thus Hoeffding's inequality can be used to quantify the value of m required to guarantee low approximation error.

However, since each such randomized approximation step has some non-zero probability of failing to compute an accurate backup value, the algorithm employs additional mechanism to force the total number of approximations to be bounded, and so we may use a union bound to bound the total failure probability of the whole algorithm. This goal is achieved by introducing a timestamp for each state–action pair to avoid unnecessary Bellman backups. The complete algorithm, which we describe in Algorithm 27, is slightly simplified to get rid of an unnecessary exploration bonus in the original presentation by Strehl et al. [2006b].

Algorithm 27 Randomized RTDP.

Inputs: $\mathcal{S}, \mathcal{A}, \gamma, \mathfrak{O}, \epsilon_1 \in \mathbb{R}_+, m \in \mathbb{N}$.
for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ **do**
  $Q(s,a) \leftarrow V_{\max}$ {optimistic initialization of state–action values}
  $A(s,a) \leftarrow 0$ {timestamp of last attempted update}
end for
$t^* \leftarrow 0$ {timestamp of most recent Q-value change}
for $t = 1, 2, 3, \ldots$ **do**
  Observe the current state s_t, take action $a_t \leftarrow \text{argmax}_{a \in \mathcal{A}} Q(s_t, a)$, obtain reward r_t, and transition to a next state s_{t+1}.
  if $A(s,a) \leq t^*$ **then**
    Sample m independent transitions from the generative model $\mathfrak{O}$: $\mathcal{D} = \{(r_{t1}, s'_{t1}), (r_{t2}, s'_{t2}), \ldots, (r_{tm}, s'_{tm})\}$ and compute an attempted update:

$$q \leftarrow \frac{1}{m} \sum_{i=1}^{m} \left(r_{ti} + \gamma \max_{a \in \mathcal{A}} Q(s'_{ti}, a) \right).$$

    if $q \leq Q(s_t, a_t) - \epsilon_1$ **then**
      $Q(s_t, a_t) \leftarrow q$.
      $t^* \leftarrow t$
    end if
    $A(s,a) \leftarrow t$.
  end if
end for

Theorem 23 *[Strehl et al. 2006b] When $\epsilon_1 = \epsilon(1-\gamma)/8$, and*

$$m = \mathcal{O}\left(\frac{(\gamma V_{\max}+1)^2}{\epsilon^2(1-\gamma)^2}\ln\frac{|\mathcal{S}|\,|\mathcal{A}|}{\epsilon(1-\gamma)\delta}\right),$$

randomized RTDP is PAC-MDP with the following sample complexity of exploration:

$$\mathcal{O}\left(\frac{V_{\max}}{\epsilon(1-\gamma)}\left(\frac{|\mathcal{S}|\,|\mathcal{A}|\,V_{\max}}{\epsilon(1-\gamma)}+\ln\frac{1}{\delta}\right)\ln\frac{1}{\epsilon(1-\gamma)}\right).$$

The proof of the theorem follows a similar line of reasoning to that of Theorem 22. We will need the following definitions and lemmas, which are helpful for understanding the behavior of randomized RTDP as well. Proofs for the lemmas are found in §8.5.2.

Definition 16 *In the execution of randomized RTDP, an* attempted update *occurs when Line 9 of Algorithm 27 is executed. A* successful update *occurs at timestep t when the state–action value function changes in Line 11, namely, when $q \le Q_t(s_t,a_t) - \epsilon_1$.*

Definition 17 *We define A_{RTDP} as the event that the following holds for all timesteps t, in which an attempted update occurs:*

$$|q - \mathfrak{B}Q_t(s_t,a_t)| \le \epsilon_1. \tag{8.1}$$

Lemma 39 *During an entire execution of randomized RTDP, there are at most*

$$\kappa_s \stackrel{\text{def}}{=} \frac{|\mathcal{S}|\,|\mathcal{A}|\,V_{\max}}{\epsilon_1}$$

successful updates and

$$\kappa_a \stackrel{\text{def}}{=} |\mathcal{S}|\,|\mathcal{A}|\,(1+\kappa_s)$$

attempted updates.

Lemma 40 *It suffices to set*

$$m = \frac{(\gamma V_{\max}+1)^2}{2\epsilon_1^2}\ln\frac{4\kappa_a}{\delta}$$

to guarantee that event A_{RTDP} holds with probability at least $1-\delta/2$.

Lemma 41 *Assuming event A_{RTDP} holds, then the following holds true for an entire execution of randomized RTDP: If an attempted update occurs at timestep t and $(s_t,a_t) \notin K_t$, then the update will be successful.*

Lemma 42 *Assuming event A_{RTDP} holds, then we have at all timesteps t that*

$$Q_t(s,a) \geq Q^*(s,a) - \frac{\epsilon_1}{1-\gamma} = Q^*(s,a) - \frac{\epsilon}{8}.$$

We are now ready to prove the main theorem of this subsection.

PROOF (of Theorem 23). The proof relies on Theorem 4 by verifying that three conditions hold. First, assume A_{RTDP} holds. The optimism and accuracy conditions are verified by Lemmas 42 and 37, respectively. The bounded-surprises condition also holds with $\zeta(\epsilon,\delta) = \kappa_s$, because (i) Lemma 41 guarantees that a successful update occurs whenever a state–action pair outside K_t is experienced, and (ii) any change in the value function coincides, by definition, to successful updates.

If, on the other hand, A_{RTDP} fails, then the three conditions may not be satisfied. However, the probability that A_{RTDP} fails is at most $\delta/2$, according to Lemma 40, which completes the proof. □

8.2 Delayed Q-learning

Delayed Q-learning [Strehl et al. 2006c] is the first model-free PAC-MDP algorithm for general finite MDPs. It generalizes earlier results for deterministic finite MDPs [Koenig and Simmons 1996] and bears a number of similarities to randomized RTDP. Some of its variants are developed by Strehl [2007b], which use techniques such as interval estimation. This section presents a slightly improved version that does not use the somewhat unnatural exploration bonus from the original description by Strehl et al. [2006c], and shows the algorithm is PAC-MDP using Theorem 4 and Lemma 37.

8.2.1 Algorithm

Delayed Q-learning may be viewed as a modified version of randomized RTDP that does not require a generative model. Instead, the agent only observes one sample transition for the action it takes in the current state, as described in the online interaction protocol (Definition 1). This challenge calls for corresponding changes in the algorithm. A complete pseudocode description is in Algorithm 28. Below, we give informal explanations for why the algorithm is PAC-MDP. A formal analysis is given in the next subsection.

First, like randomized RTDP, delayed Q-learning uses optimistic initialization of the value function, and waits until m transitions from (s, a) are gathered before considering an update of $Q(s, a)$. When m is sufficiently large (but is still bounded by a polynomial), the new value of $Q(s, a)$ is still optimistic (modulo a small gap of $\Theta(\epsilon)$), based on the sample-complexity result for the subinterval prediction problem in §5.3.4. Thus, the optimism condition in Theorem 4 is satisfied.

Second, we use Definition 15 for the known state–action set, K_t, similar to the analysis of randomized RTDP. By virtue of Lemma 37, the accuracy condition of Theorem 4 is automatically satisfied.

Third, delayed Q-learning maintains a learning flag for each state–action pair that is TRUE when this pair does not belong to the set K_t. However, an unsuccessful attempted update in $Q(s, a)$ at timestep t does not necessarily imply with high probability that $(s, a) \notin K_t$, which is different from randomized RTDP. The reason is that the state–action value function may change during the collection of the m samples; consequently, it is possible that $(s, a) \in K_{k_1}$ but $(s, a) \notin K_{k_m}$, where $k_1 < k_2 < \cdots < k_m$ are the m most recent timesteps in which (s, a) is experienced. Luckily, by using a slightly more complicated learning-flag mechanism, we can still bound the total number of attempted updates and consequently the total number of changes in K_t. This fact is used to satisfy the bounded-surprises condition of Theorem 4.

Finally, we note that the batch update achieved by averaging m samples is equivalent to performing stochastic gradient descent using these samples with a sequence of learning rates: $1, \frac{1}{2}, \ldots, \frac{1}{m}$, assuming updates on $Q(s, a)$ do not affect the sequence of samples. This observation justifies the name of the algorithm, emphasizing both the relation to Q-learning with the harmonic learning rates above and the batch nature of the updates.

8.2.2 Sample Complexity of Exploration

The main result of this section is the following theorem, which says that delayed Q-learning is PAC-MDP.

Algorithm 28 Delayed Q-learning

Inputs: $\mathcal{S}, \mathcal{A}, \gamma, m \in \mathbb{N}, \epsilon_1 \in \mathbb{R}_+$
for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ **do**
  $Q(s,a) \leftarrow V_{\max}$ {optimistic initialization of state–action values}
  $U(s,a) \leftarrow 0$ {used for attempted updates of the value in (s,a)}
  $B(s,a) \leftarrow 0$ {beginning timestep of attempted update in (s,a)}
  $C(s,a) \leftarrow 0$ {counter for (s,a)}
  $L(s,a) \leftarrow$ TRUE {the learning flag}
end for
$t^* \leftarrow 0$ {timestamp of most recent state–action value change}
for $t = 1, 2, 3, \cdots$ **do**
  Observe the current state s_t, take action $a_t \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} Q(s_t, a)$, receive reward $r_t \in [0,1]$, and transition to a next state s_{t+1}.
  if $B(s_t, a_t) \leq t^*$ **then**
    $L(s_t, a_t) \leftarrow$ TRUE
  end if
  if $L(s_t, a_t) =$ TRUE **then**
    if $C(s_t, a_t) = 0$ **then**
      $B(s_t, a_t) \leftarrow t$
    end if
    $C(s_t, a_t) \leftarrow C(s_t, a_t) + 1$
    $U(s_t, a_t) \leftarrow U(s_t, a_t) + r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a)$
    if $C(s_t, a_t) = m$ **then**
      $q \leftarrow U(s_t, a_t)/m$
      if $Q(s_t, a_t) - q \geq \epsilon_1$ **then**
        $Q(s_t, a_t) \leftarrow q$
        $t^* \leftarrow t$
      else if $B(s_t, a_t) > t^*$ **then**
        $L(s_t, a_t) \leftarrow$ FALSE
      end if
      $U(s_t, a_t) \leftarrow 0$
      $C(s_t, a_t) \leftarrow 0$
    end if
  end if
end for

Theorem 24 *In delayed Q-learning, if parameters are set by*

$$m = \mathcal{O}\left(\frac{{V_{\max}}^2}{\epsilon^2(1-\gamma)^2}\ln\frac{|\mathcal{S}|\,|\mathcal{A}|}{\epsilon\delta(1-\gamma)}\right),$$

and

$$\epsilon_1 = \frac{\epsilon(1-\gamma)}{8},$$

then the algorithm is PAC-MDP with the following sample complexity of exploration:

$$\mathcal{O}\left(\frac{|\mathcal{S}|\,|\mathcal{A}|\,{V_{\max}}^4}{\epsilon^4(1-\gamma)^4}\ln\frac{1}{\epsilon(1-\gamma)}\ln\frac{|\mathcal{S}|\,|\mathcal{A}|}{\epsilon\delta(1-\gamma)}\right).$$

The proof of this theorem slightly improves the original one by Strehl et al. [2006c]. We will need a modified version of Definition 16 and a few technical lemmas, whose proofs are found in §8.5.3.

Definition 18 *In the execution of delayed Q-learning, an* attempted update *occurs when the condition in Line 20 of Algorithm 28 is true (and thus Lines 21–29 are executed). A* successful update *occurs when the state–action value function is changed in Line 23. An attempted update that is not successful is called an* unsuccessful update.

Lemma 43 *During an entire execution of delayed Q-learning, there are at most*

$$\kappa_s \stackrel{\text{def}}{=} \frac{|\mathcal{S}|\,|A|\,V_{\max}}{\epsilon_1}.$$

successful updates and

$$\kappa_a \stackrel{\text{def}}{=} |\mathcal{S}|\,|\mathcal{A}|\,(1+\kappa_s).$$

attempted updates.

Definition 19 *We define A_{DQL} as the following event: for all timesteps t in which an attempted update occurs, if $(s_t, a_t) \notin K_{k_1}$, then the update will succeed, where $k_1 < k_2 < \cdots < k_m = t$ are the m last timesteps during which (s_t, a_t) is experienced.*

Lemma 44 *It suffices to set*

$$m = \frac{(1+\gamma V_{\max})^2}{2{\epsilon_1}^2}\ln\frac{3\kappa_a}{\delta} \tag{8.2}$$

in delayed Q-learning to guarantee event A_{DQL} holds with probability at least $1-\delta/3$.

The next lemma states that, with high probability, delayed Q-learning will maintain an optimistic value function.

Lemma 45 *During execution of delayed Q-learning, if m satisfies Equation 8.2, then $Q_t(s,a) \geq Q^*(s,a) - \epsilon_1/(1-\gamma)$ holds for all timesteps t and state–action pairs (s,a), with probability at least $1 - \delta/3$.*

Lemma 46 *If event A_{DQL} occurs, then the following holds: If an unsuccessful update occurs at time t and $L(s_t, a_t)$ is set to* FALSE*, then $(s_t, a_t) \in K_{t+1}$.*

The following lemma bounds the number of timesteps t in which a state–action pair $(s,a) \notin K_t$ is experienced.

Lemma 47 *If event A_{DQL} occurs, then the number of timesteps t such that $(s_t, a_t) \notin K_t$ is at most*

$$\frac{2mV_{\max}|\mathcal{S}||\mathcal{A}|}{\epsilon_1}.$$

Proof (of Theorem 24). The proof relies on Theorem 4 by verifying that three conditions hold. We have shown, with high probability that, the optimism, accuracy, and bounded-surprises conditions hold in Lemmas 45, 37, and 47, respectively. Using a union bound, the total probability that any of them fails to hold is at most δ, thus completing the proof. □

8.2.3 An Extension to State Abstractions

Delayed Q-learning is stated for finite MDPs and the sample complexity bound in Theorem 24 is in terms of the number of states and actions. However, it is possible to extend the algorithm and analysis to infinite-state MDPs when a certain type of state abstraction (§3.2.1) is used.

In MDP state abstractions, a function ϕ is used to aggregate "similar" states into one abstract state. Recall that the Q^*-irrelevance abstraction, ϕ_{Q^*}, aggregates states with the same optimal Q-values. In the context of our discussion, we can relax the notion a little by allowing a small gap in the abstraction. We call an abstraction function ϕ

a Q^*-*irrelevance abstraction with gap* ε if the following holds: for all $s_1, s_2 \in \mathcal{S}$ and $a \in \mathcal{A}$: $\phi(s_1) = \phi(s_2)$ implies $|Q^*(s_1, a) - Q^*(s_2, a)| \leq \varepsilon$. Given such an abstraction function ϕ, the original MDP can be turned into an abstract MDP, $\bar{M} = \langle \bar{\mathcal{S}}, \mathcal{A}, \bar{T}, \bar{R}, \gamma \rangle$.[1] Furthermore, we assume that $\left|\bar{\mathcal{S}}\right|$ is finite.

The modification to the algorithm is straightforward. We simply run delayed Q-learning on the abstract MDP $\bar{M}$ and estimate the optimal abstract Q-function, $\bar{Q}^*(\bar{s}, a)$, for $\bar{s} \in \bar{\mathcal{S}}$. Thus, even if $\mathcal{S}$ is infinitely large, we may still represent $\bar{Q}^*(s, a)$ by a finite-entry table.

To analyze the sample complexity of exploration, we shall assume the abstract Q-function allows a sufficiently accurate representation of the true Q-function; otherwise, a policy derived from the abstract Q-function may be poor in the true MDP. Specifically, we assume that ϕ is a Q^*-irrelevance abstraction with gap $\varepsilon = \mathcal{O}(\epsilon(1-\gamma))$. Consequently, every attempted update value (*i.e.*, the quantity q computed in Line 21 of Algorithm 28) may introduce an additional error term on the order of $\mathcal{O}(\epsilon(1-\gamma))$,[2] which in turn translates into an additional $\mathcal{O}(\epsilon)$ gap on the right-hand side of the bound in Lemma 45. This additional term does not violate the accuracy requirement in Theorem 4 if we re-scale the error parameter ϵ, and so the same asymptotic sample complexity of exploration holds. The rest of the analysis in §8.2.2 remains unchanged. Thus, we have proved the following corollary of Theorem 24.

Corollary 1 *When provided with a* Q^**-irrelevance abstraction* ϕ *with gap* $\mathcal{O}(\epsilon(1-\gamma))$*,* delayed Q-learning *is PAC-MDP with the following sample complexity of exploration:*

$$\mathcal{O}\left(\frac{\left|\bar{\mathcal{S}}\right| |\mathcal{A}| \, {V_{\max}}^4}{\epsilon^4 (1-\gamma)^4} \ln \frac{1}{\epsilon(1-\gamma)} \ln \frac{\left|\bar{\mathcal{S}}\right| |\mathcal{A}|}{\epsilon\delta(1-\gamma)}\right).$$

8.2.4 Optimality of Delayed Q-learning

The main result of this section (Theorem 25) is an improvement on previous sample complexity of exploration lower bounds for reinforcement learning in MDPs. Existing

[1] In defining the abstract MDP, any well-defined weighting function can be used (*c.f.*, Definition 4).

[2] Equivalently, every attempted update involves solving an agnostic version of the subinterval-prediction problem, in which the best prediction has an error of $\mathcal{O}(\epsilon(1-\gamma))$.

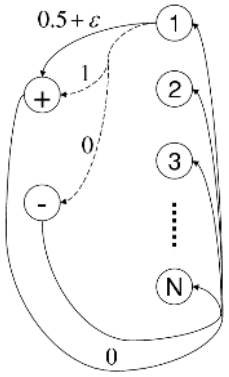


Figure 8.1: The difficult-to-learn MDPs used to show an improved lower bound of sample complexity of exploration.

results [Kakade 2003] show a linear dependence on $|\mathcal{S}|$ and ϵ, but we find that a linearithmic dependence on $|\mathcal{S}|$ and a quadratic dependence on ϵ are necessary for *any* deterministic reinforcement-learning algorithm. We note that our analysis has some similarity to the lower-bound analysis of Leffler et al. [2005], although their result is for a different learning model.

Theorem 25 *For any deterministic reinforcement-learning algorithm* $\mathbf{A}$*, there is an MDP* $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ *such that the sample complexity of exploration of* $\mathbf{A}$ *in* M *is*

$$\Omega\left(\frac{|\mathcal{S}|\,|\mathcal{A}|}{\epsilon^2} \ln \frac{|\mathcal{S}|}{\delta}\right). \tag{8.3}$$

To prove this theorem, consider the family of MDPs depicted in Figure 8.1. Each MDP in this family has $N+2$ states: $\mathcal{S} = \{1, 2, \ldots, N, +, -\}$, and $A = |\mathcal{A}|$ actions. For convenience, denote by $[N]$ the set $\{1, 2, \ldots, N\}$. Transitions from each state $i \in [N]$ are the same, so only the transitions from state 1 are depicted. One of the actions (the solid one) deterministically transports the agent to state $+$ with reward $0.5 + \epsilon$. Let a be any of the other $A - 1$ actions (the dashed ones). From any state $i \in [N]$, taking a will cause a transition to $+$ with reward 1 and probability p_{ia}, and to $-$ with reward 0 otherwise, where $p_{ia} \in \{0.5, 0.5 + 2\epsilon\}$ is a number very close to $0.5 + \epsilon$. Furthermore, for each $i \in [N]$, there is at most one action a such that $p_{ia} = 0.5 + 2\epsilon$. Transitions from states $+$ and $-$ are identical: they simply reset the agent to one of the states in $[N]$ uniformly at random.

In fact, the MDP defined above can be viewed as N copies of a multi-armed bandit problem where the states $+$ and $-$ are dummy states for resetting the agent to the next "real" state. Therefore, the optimal action in a state i is independent of the optimal action in any other state: it is the solid action if $p_{ia} = 0.5$ for all dashed actions a; otherwise, it is the dashed action a for which $p_{ia} = 0.5 + 2\epsilon$. Intuitively, this MDP is hard to learn for exactly the same reason that a biased coin is hard to learn if the bias (that is, the probability of head after a coin toss) is close to 0.5.

The lemma below follows from a sample-complexity lower bound stated in Lemma 11. Its proof is in the appendix.

Lemma 48 *There exists constants $c_1, c_2 \in (0, 1)$ such that during a whole run of* **A**, *for any state $i \in [N]$, the probability (with respect to the randomness of the MDP) that sub-optimal actions are taken in i more than m_i times is at least*

$$p(m_i) := c_2 \exp\left(-\frac{m_i \epsilon^2}{c_1 A}\right).$$

With this lemma, we are ready to prove the main theorem.

Proof (of Theorem 25). Let $\zeta(\epsilon, \delta)$ be an upper bound of the sample complexity of any PAC-MDP algorithm **A** with probability at least $1 - \delta$. Let sub-optimal actions be taken m_i times in state $i \in [N]$ during a whole run of **A**. Consequently,

$$\delta \geq \Pr\left(\sum_{i=1}^{N} m_i > \zeta(\epsilon, \delta)\right) = 1 - \Pr\left(\sum_{i=1}^{N} m_i \leq \zeta(\epsilon, \delta)\right),$$

where the first step is because the actual sample complexity is at least $\sum_i m_i$—every timestep in which a suboptimal action is taken counts towards the sample complexity.

We wish to find a lower bound for the last expression above by optimizing the values of m_is subject to the constraint: $\sum_i m_i \leq \zeta(\epsilon, \delta)$. Due to the statistical independence of what states $i \in [N]$ are visited by the algorithm,[3] we can factor the probability above to obtain

$$\delta \geq 1 - \max_{m_1, \ldots, m_N; \sum_i m_i \leq \zeta(\epsilon, \delta)} \prod_{i=1}^{N} (1 - p(m_i)).$$

[3]It does not help for the algorithm to base its policy in one state on samples collected in other states, due to the independence of states in this MDP. If an algorithm attempts to do so, an adversary can make use of this fact to assign p_{ia} to even *increase* the failure probability of the algorithm.

where Lemma 48 is applied.

We now use Lemma 25 to obtain a lower bound of the last expression above, which in turn lower-bounds δ. Applying this lemma with $c = c_2$ and $\Delta = \exp(-\frac{\epsilon^2}{c_1 A})$ gives

$$\delta \geq 1 - \left(1 - c_2 \exp\left(-\frac{\zeta(\epsilon, \delta)\epsilon^2}{c_1 N A}\right)\right)^N. \tag{8.4}$$

The theorem then follows immediately from Lemma 26 with $\eta = \epsilon^2/A$ using the fact that $N = |\mathcal{S}| - 2$. □

8.3 LSPI-Rmax

In this section, we illustrate how theoretical insights in provably PAC-MDP analysis may result in more practical algorithms with good empirical performance in balancing exploration and exploitation. Specifically, we describe a variant of the model-free LSPI algorithm (§4.2.3) that is motivated by ideas in KWIK-Rmax and is thus called LSPI-Rmax [Li et al. 2009a]. Although the algorithm shares some similarity to Rmax-style algorithms and is reduced to exactly Rmax in a special case, we view it as a model-free algorithm since the MDP model is not estimated explicitly in general in LSPI-Rmax.

8.3.1 Algorithm

The algorithm uses linear function approximation (§3.2.2) to represent a value function:

$$Q_{\mathbf{w}}(s, a) \stackrel{\text{def}}{=} \mathbf{w}^\top \boldsymbol{\phi}(s, a),$$

where $\mathbf{w} \in \mathbb{R}^k$ is a k-vector and $\boldsymbol{\phi} : \left(\mathbb{R}^k\right)^{\mathcal{S}\times\mathcal{A}}$ consists of k predefined features.

As a first step toward combining LSPI with Rmax-style exploration, we extend the concept of known state–action pairs as used by KWIK-Rmax to continuous state spaces. Our approach relies on the intuition that the dynamics of a state–action pair can be reliably predicted with sufficient sample transitions from its neighbor state–actions, similar to the idea in locally weighted regression [Atkeson et al. 1997a]. Formally, the algorithm requires a distance function that computes the dissimilarity (or distance) between two state–action pairs: $d : (\mathcal{S} \times \mathcal{A})^2 \to \mathbb{R}_+$.

Definition 20 *Let $\varepsilon > 0$ and $m > 0$ be two predefined constants. Given a set of sample transitions $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}$, a state–action pair (s, a) is called* (ε, m)-known *if*

$$\left|\{(s_i, a_i, r_i, s'_i) \in \mathcal{D} \mid d(s, a, s_i, a_i) \leq \varepsilon\}\right| \geq m.$$

In other words, $\mathcal{D}$ contains at least m samples that are ε-close to (s, a). Furthermore, a state s is called (ε, m)-known if (s, a) is (ε, m)-known for all actions a.

Using this notion, we are now ready to present LSTDQ-Rmax (Algorithm 29), a variant of LSTDQ (*c.f.*, §4.2.3) that assigns $V_{\max}$ to unknown states and unknown state–actions. Let (s, a, r, s') be a sample in $\mathcal{D}$, and consider the following cases:

- If both (s, a) and s' are (ε, m)-known, LSTDQ-Rmax operates exactly like LSTDQ, as in Lines 7—9 of Algorithm 29.
- If (s, a) is (ε, m)-known but s' is not, then we pretend s' is a goal state whose value is the largest possible, $V_{\max}$. Therefore, the return of (s, a) is the sum of the immediate reward, r, and the pretended *discounted* value of s', $\gamma V_{\max}$; this corresponds to Lines 10—12 that encourage the agent to further explore the neighborhood of s'.
- Similarly, when (s, a) is not (ε, m)-known, the return of this state–action pair is treated as $V_{\max}$, which results in Lines 15—16.
- Finally, since solutions found by LSTDQ-Rmax are biased by the sample distribution in the input sample set $\mathcal{D}$, Lines 18—21 add artificial samples to (s, a') pairs so that untried actions a' in s are preferred when a nearby state is visited in the future. These state–action pairs also have the largest possible value, $V_{\max}$.

With LSTDQ-Rmax as a building block, we can give a generic form of the full online learning algorithm, LSPI-Rmax, in Algorithm 30. In this algorithm, the agent always chooses greedy actions with respect to its current value-function estimate. When new training samples are added to the sample set $\mathcal{D}$, it runs LSPI to update its value function, in which it uses LSTDQ-Rmax instead of LSTDQ to evaluate value functions.

To summarize: LSTDQ-Rmax is just like LSTDQ, except that it assigns maximum value, $V_{\max}$, to states and state–action pairs that are not (ε, m)-known; LSTDQ-Rmax

becomes LSTDQ when $m = 0$. LSPI-Rmax is just an online version of LSPI that uses LSTDQ-Rmax instead of LSTDQ to approximate the policy-evaluation step.

Algorithm 29 LSTDQ-Rmax

Input: $\mathcal{D}, \pi, \boldsymbol{\phi}, k, \gamma, \varepsilon, m$.
Output: weight vector $\mathbf{w}$ so that $Q_{\mathbf{w}} \approx Q^{\pi}$
$A \leftarrow O_{k \times k}$ (the $k \times k$ zero matrix)
$\mathbf{b} \leftarrow \mathbf{0}_k$ (the $k \times 1$ zero column-vector)
for all $(s, a, r, s') \in \mathcal{D}$ **do**
  if (s, a) is (ε, m)-known **then**
    if s' is (ε, m)-known **then**
      $A \leftarrow A + \boldsymbol{\phi}(s, a) \cdot \left(\boldsymbol{\phi}(s, a) - \gamma \boldsymbol{\phi}(s', \pi(s'))\right)^{\top}$
      $\mathbf{b} \leftarrow \mathbf{b} + \boldsymbol{\phi}(s, a) \cdot r$
    else
      $A \leftarrow A + \boldsymbol{\phi}(s, a) \cdot \boldsymbol{\phi}(s, a)^{\top}$
      $\mathbf{b} \leftarrow \mathbf{b} + \boldsymbol{\phi}(s, a) \cdot (r + \gamma V_{\max})$
    end if
  else
    $A \leftarrow A + \boldsymbol{\phi}(s, a) \cdot \boldsymbol{\phi}(s, a)^{\top}$
    $\mathbf{b} \leftarrow \mathbf{b} + \boldsymbol{\phi}(s, a) \cdot V_{\max}$
  end if
  for all $a' \in A \setminus \{a\}$ where (s, a') is not (ε, m)-known **do**
    $A \leftarrow A + \boldsymbol{\phi}(s, a') \cdot \boldsymbol{\phi}(s, a')^{\top}$
    $\mathbf{b} \leftarrow \mathbf{b} + \boldsymbol{\phi}(s, a') \cdot V_{\max}$
  end for
end for
Return $\mathbf{w} = A^{-1}\mathbf{b}$

Algorithm 30 LSPI-RMAX

Input: $\boldsymbol{\phi}, k, \gamma, \varepsilon, m$.
Initialize $\mathbf{w}$, s_1, and set $\mathcal{D} \leftarrow \emptyset$
for $t = 1, 2, 3, \cdots$ **do**
  Choose the greedy action a_t in s_t with respect to $Q_{\mathbf{w}}$, observe reward r_t and s_{t+1}
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
  Update $\mathbf{w}$ by running LSPI with LSTDQ-Rmax instead of LSTDQ.
end for

8.3.2 Implementation Issues

While we have given the generic form of LSPI-Rmax, a number of implementation issues remain open. We describe our solutions to these problems, but note that other solutions do exist and might work better in some situations.

The first natural question is how to decide whether a state–action pair is (ε, m)-known or not. A straightforward algorithm would be to search over all samples in $\mathcal{D}$ and count the number of nearest neighbors within ε distance. However, when $\mathcal{D}$ is large, even this $\mathcal{O}(|\mathcal{D}|)$-time algorithm is far too expensive. A better way is to employ smarter nearest-neighbor search techniques such as kd-trees [Friedman et al. 1977] whose time complexity is sub-linear in $|\mathcal{D}|$, but still is prohibitively expensive in high dimension state spaces. Therefore, it might be worthwhile to sacrifice exact counting for faster computation. In our implementation, we *coarsely* discretize $\mathcal{S} \times \mathcal{A}$ into bins, and the number of nearest neighbors is approximated by counting how many samples in $\mathcal{D}$ fall in the same bin. Thus, all samples in a bin are simultaneously known or simultaneously unknown under this approximation.[4] By maintaining a counter for each bin, we are able to achieve a much lower computational complexity that is linear in the state dimension. Therefore, this binning implementation can be viewed as a computationally efficient approximation to the exact nearest-neighbor search procedure such as kd-tree algorithms. We note that better choices for identifying known state–actions are possible in the presence of domain knowledge, and the number of bins may grow sub-exponentially. Another useful idea is to use variable-resolution bins, such as those used by Nouri and Littman [2009].

It is important to note that the discretization procedure we use here should not be confused with discretization for solving large MDPs. Discretization there directly decides the complexity of value functions being considered, and solving a discretized model often requires repeated access to the whole model (*e.g.*, in the dynamic-programming algorithms surveyed in §3.1). Here, in contrast, discretization is used only to indicate whether enough samples have been observed in a local region, that is, whether a state–action pair is known. Thus, the class of value functions under consideration and the computational complexity mostly depend on the number of features, rather than number of bins used by LSPI-Rmax. These differences make our approach less prone to the curse of dimensionality than discretization methods.

[4]This implementation also has a desired side-effect of smoothing the optimal value function of the known-state MDP, which makes it easier to learn the optimal value function and policy for the known-state MDP.

A second problem is that $\mathcal{D}$ keeps growing without limit as time goes on. However, if the MDP dynamics are smooth[5] (which is often the case in practice), it is unnecessary to keep all samples in $\mathcal{D}$ if many of them are close to each other. Thus, we have chosen to avoid adding a sample to $\mathcal{D}$ if the corresponding bin size reaches a predefined capacity, which varied between 50 and 100 in our experiments in §8.3.4. Excluding samples has the effect of making the sample distribution in $\mathcal{D}$ more uniform, which is often preferred for the policy-improvement step of LSPI in practice [Lagoudakis and Parr 2003a].

Third, Lines 18—21 ensure that untried actions a' in s are preferred when a nearby state is visited in the future by using artificial samples. Empirically, it is desirable to avoid adding too many such artificial samples, which may overwhelm the real ones. There are many ways to do so—in our implementation, an artificial sample is used only when the bin corresponding to (s, a') is empty. This choice has the desired property that if the local region of (s, a') is underrepresented, the artificial sample has the effect of increasing the value of a' in s so as to encourage exploration of a' in the neighborhood of s; on the other hand, if a' has been tried in nearby states of s', the algorithm will use those real samples and does not use artificial ones.

Finally, LSPI is invoked in each step in Algorithm 30. Since LSPI is rather expensive (the complexity of each iteration is on the order of $\mathcal{O}(k^3)$) and adding a sample to the training set $\mathcal{D}$ usually has small effects on the value function it finds, our implementation calls LSPI (Line 6 in Algorithm 30) only after a certain number of samples are added to $\mathcal{D}$.

8.3.3 Relation to Rmax

Because LSPI-Rmax's root is in Rmax, it is natural to consider the relation between them. Given a finite MDP, Rmax estimates $Q(s, a)$ for each (s, a). If we view this tabular representation as a linear function approximation with indicator features (§3.2.2), then LSPI-Rmax precisely duplicates Rmax with $\varepsilon = 0$. (Note that Rmax has a similar parameter m.) In fact, what LSTDQ-Rmax solves for is the state–action value function

[5]Nearby states have similar transition and reward functions if the MDP's dynamics are smooth.

of policy π on the empirical known state–state MDP $\hat{M}_K$ maintained by Rmax. Consequently, Line 6 of Algorithm 30 becomes exact policy iteration and returns the optimal state–action value function of $\hat{M}_K$.

This connection to Rmax serves as a sanity check of the plausibility of our approach. We can make this similarity formal. A thorough analysis of LSPI-Rmax is difficult, since a nontrivial performance guarantee for LSPI itself is open except in limited situations [Lagoudakis and Parr 2003a; Antos et al. 2008]. Our preliminary analysis is specific to the bin-based implementation described in the previous section. Similar to known states, we call a bin to be *known* if every action has been tried at least m times in states in this bin. Suppose the state space is partitioned into C disjoint bins. Let $K \subseteq \{1, 2, \cdots, C\}$ be the subset of *known* bins, then the *known-bin MDP* can be defined by $M_K \stackrel{\text{def}}{=} \langle \mathcal{S}, \mathcal{A}, T_K, R_K, \gamma \rangle$ where

$$R_K(s,a) \stackrel{\text{def}}{=} \begin{cases} R(s,a) & \text{if } s \text{ is in a known bin} \\ 1 & \text{otherwise,} \end{cases}$$

$$T_K(s,a,s') \stackrel{\text{def}}{=} \begin{cases} T(s,a,s') & \text{if } s \text{ is in a known bin} \\ \mathbb{I}(s = s') & \text{otherwise.} \end{cases}$$

Assumption 3 below asserts that LSPI, when using a sufficiently good feature function $\boldsymbol{\phi}$, is able to return a near-optimal policy in known-bin MDP.

Assumption 3 *Let $\epsilon, \delta \in (0,1)$ be given. Let $m = m(\epsilon, \delta)$ be some positive constant depending on ϵ and δ. We call C a* cover number *of the MDP if the state space $\mathcal{S}$ can be partitioned into C disjoint bins: $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \cdots \cup \mathcal{S}_C$, such that, with probability at least $1 - \delta$, LSPI with the given features $\boldsymbol{\phi}$ returns an ϵ-accurate value function in the known-state MDP M_K for any $K \subseteq \{1, 2, \ldots, C\}$.*

This assumption allows one to show a bound on the sample complexity of exploration stated in Theorem 26, whose proof follows similar steps as the sample complexity proof for KWIK-Rmax. Although it is hard to verify Assumption 3 in practice, the theorem serves as a best-case sanity check that the algorithm does employ the exploration

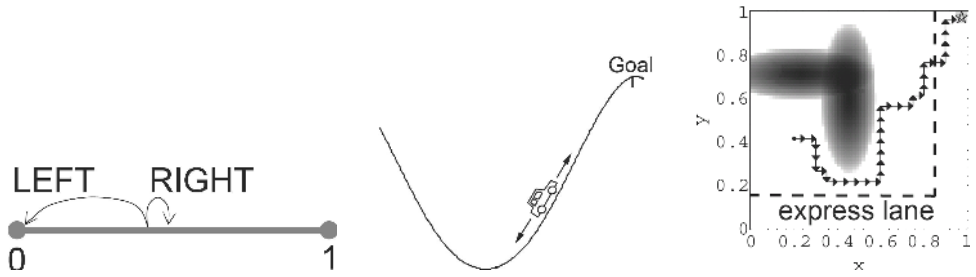


Figure 8.2: Problems used in experiments. Left: ContCombLock. Middle: MountainCar (adapted from Sutton and Barto [1998]). Right: ExpressWorld (adapted from Boyan and Moore [1995]).

efficiency of Rmax. An important open question is to generalize the analysis to broader classes of linear function approximation.

Theorem 26 *When it is implemented using bins $S_1, \ldots, S_C$ that are defined in Assumption 3, LSPI-Rmax's sample complexity of exploration is*

$$\mathcal{O}\left(\frac{V_{\max}}{\epsilon(1-\gamma)}\left(CAm(\epsilon(1-\gamma), \delta/(CA)) + \ln\frac{1}{\delta}\right)\frac{1}{\epsilon(1-\gamma)}\right).$$

LSPI-Rmax is similar to metric E^3 in the sense that they both depend on some kind of smoothness assumption. The notation of (ε, m)-known state–actions plays a similar role to the local modeling assumption. Metric E^3 estimates the model explicitly based on samples and then employs a hypothetical planner to get the desired policy. In contrast, LSTDQ-Rmax builds a compressed empirical model implicitly [Boyan 2002; Parr et al. 2008] and approximate policy iteration is then used to solve the planning problem. The near-optimality part in Assumption 3 is roughly the analog of the approximate planner assumption made in metric E^3. Since LSPI does not make a clean distinction between learning and planning, it is not clear how to use the same assumptions that applied to metric E^3 to analyze LSPI-Rmax.

8.3.4 Experiments

This section reports experimental results on four continuous, episodic domains ordered by increasing difficulty of exploration. We did not try to optimize features, which is an

important problem. Further investigation is needed to study the effect of features on exploration.

Domains

MountainCar [Sutton and Barto 1998] is a problem in which the agent tries to drive a car to a hilltop (Figure 8.2(a)). To do so, the agent has to reverse the car to move away from the goal and then apply full throttle until it reaches the hilltop. This problem has two continuous state variables and three actions. Every step gives rise to a reward of -1, and the state transition is governed by a system of nonlinear equations [Sutton and Barto 1998]. We used CMAC features consisting of 3 layers of 4×4 tilings, which are then repeated for each of the three actions, leading to a total of $3 \times 3 \times 4 \times 4 = 144$ features. The same approach of repeating features for every action was adopted in the other problems.

Bicycle is the problem of balancing a simulated bicycle. We used the deterministic version of bicycle [Randløv and Alstrøm 1998] (*i.e.*, the noise in the displacement action is always zero) to illustrate how LSPI-Rmax works in a challenging, high dimensional problem, although the problem is somewhat easier than the original, stochastic one. There are six continuous state variables and two continuous action variables. Each balancing step leads to a reward of $+1$, and an episode terminates when the bicycle falls. We used the same set of hand-coded features and the same discretization in the continuous action space as in the original LSPI paper [Lagoudakis and Parr 2003a]; that is, there were 100 features and 5 actions.

ExpressWorld is adapted from PuddleWorld [Sutton 1996]. In PuddleWorld, the state space is a two-dimensional continuous grid world in which the agent can move along four directions (N, E, S, and W) to reach the goal region in the northeast corner while trying to avoid two puddles (Figure 8.2(b)). Each step yields a -1 reward plus a penalty for entering the puddle region. To make exploration more important in this task, we add an "express lane" 0.15 units wide—if the agent moves within this lane, every immediate reward is -0.5 instead of -1. Start states are drawn randomly from the left half plane so that the agent has to learn how to avoid puddles,

as well as to explore actively to discover the goal as well as the express lane. We have found empirically that this exploration task is quite challenging. Partly because the reward function that has sharp changes in the puddle region, it is not easy to find good features for this problem. Therefore, we simply used a 6×6 discretization of the state space and treated it as a CMAC feature with one layer of gridding, which resulted in a total of $4 \times 6 \times 6 = 144$ features.

The last problem, ContCombLock, is a continuous version of combination lock, which was designed to require a smart exploration strategy [Koenig and Simmons 1996]. The state space is a segment $[0, 1]$ with a fixed start state 0 (Figure 8.2(c)). There are two actions: left always takes the agent back to the start state 0; the other action right takes the agent from state x to a new state $y = x+0.02+\Delta$, where Δ is generated via Gaussian noise with mean 0 and standard deviation 0.005. If the agent reaches a state $x > 0.98$, the episode terminates. Every step results in a -1 reward. Therefore, the optimal policy is to always choose right, and on average each episode takes about 50 steps to finish. We used CMAC features that have 3 layers of grids, each dividing the state space into 6 pieces. Hence, $2 \times 3 \times 6 = 36$ features were used.

For Bicycle, the agent was allowed to run 2000 episodes, each of which was at most 72000 steps long; in the other problems, the agent had to run up to 200 episodes, each of which was at most 300 steps long. The discount factor was set to 0.99 for all four problems.

In the experiments, the capacity of a bin was set to 100 except in Bicycle where it was 50 (since this problem's state space is larger). We invoked LSPI in LSPI-Rmax every time 100 new samples were added to D, except in ContCombLock where LSPI was invoked immediately after a new sample was added.

We compared LSPI-Rmax against LSPI with ϵ-greedy and counter-based exploration. For each of them, we have tried different exploration parameters and report the best result. Since all problems are continuous, we had to modify the counter-based method to use the same trick as LSPI-Rmax (*cf.*, §8.3.2), and maintained a counter for each bin.

Results

Figure 8.3 shows the learning curves for cumulative reward of LSPI with different exploration rules in all four problems, where the y-axis is the cumulative rewards, averaged over 30 runs. The slope of a curve corresponds to per-episode reward. In MountainCar and Bicycle, all three exploration rules had similar cumulative rewards for the first dozen episodes, but LSPI-Rmax quickly showed its advantage over the other two and converged to a much better policy. It is also observed that ϵ-greedy and counter-based exploration rules can be better than the other, depending on specific problems. In ContCombLock, LSPI-Rmax was the only one that succeeded.

The learning curve for ExpressWorld probably best illustrates the way LSPI-Rmax works. At the beginning of learning, LSPI-Rmax actually receives much less cumulative rewards since it attempts to visit different parts of the state space and thus receives a lot of penalty for entering puddles. However, after about 10 episodes, it has obtained a better policy, which finally compensates the cost of exploration at the beginning. In contrast, LSPI with ϵ-greedy and counter-based exploration converged to suboptimal policies (visible in the graph as their reward slopes).

The success of LSPI-Rmax comes from the fact that the agent actively explores the state space. This claim is supported by Figure 8.4, which plots the states visited by the three agents in ExpressWorld for the first three episodes during a typical run. (We chose this domain for demonstration because it is easier to visualize its 2D state space). Observe that all three agents failed to reach the goal for the first three episodes. However, the LSPI-Rmax agent apparently behaves more intelligently by experiencing novel parts of the state space, while the other two agents had difficulty getting far away from the start states on the left half plane.

Finally, we study the effect of parameter m on cumulative rewards in LSPI-Rmax. Figure 8.5 plots the average per-episode reward, as well as standard deviation, of LSPI-Rmax during the entire 30 runs in ExpressWorld with different threshold m values. The results demonstrate how m controls the exploration-exploitation tradeoff. When m is small, the agent tries to exploit sooner, but risks at ending up with less effective

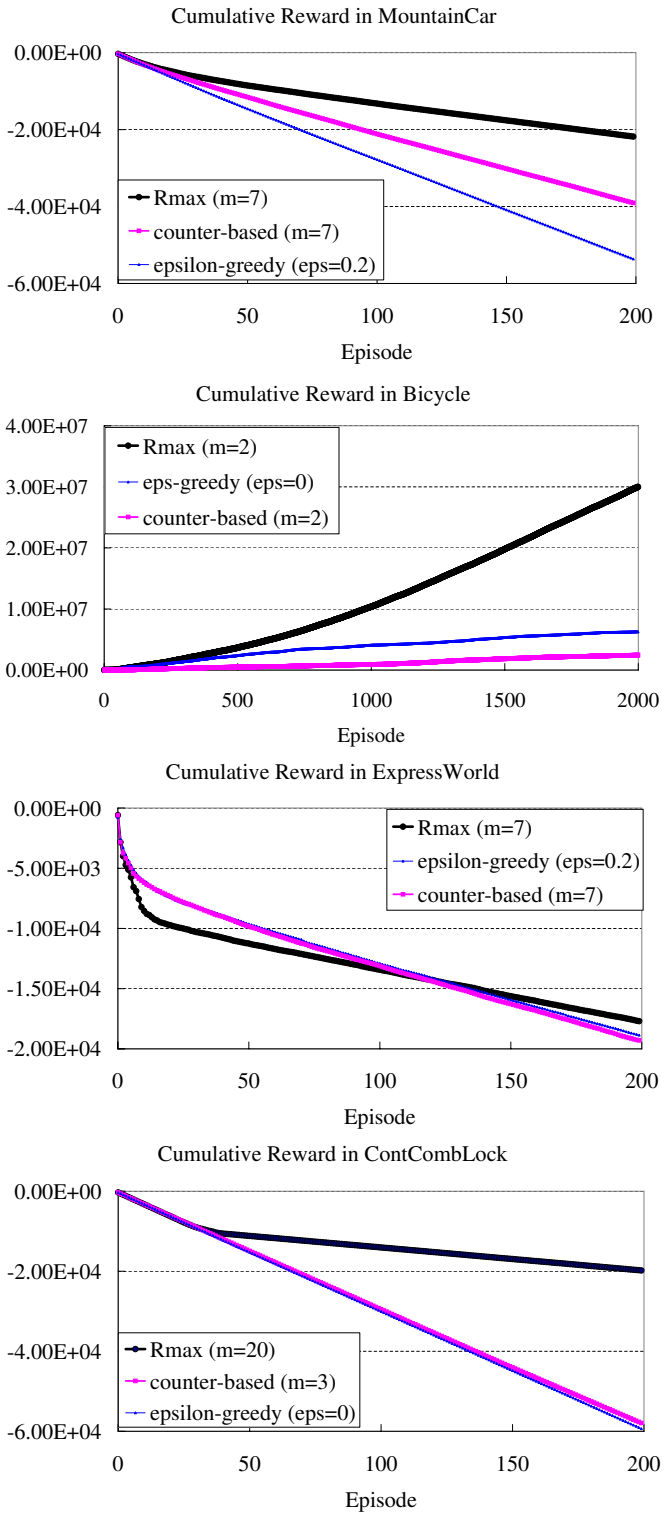


Figure 8.3: Learning curves for cumulative rewards with three exploration strategies. All results are averaged over 30 independent runs. In the last two figures, the curves for ϵ-greedy and counter-based exploration almost overlap.

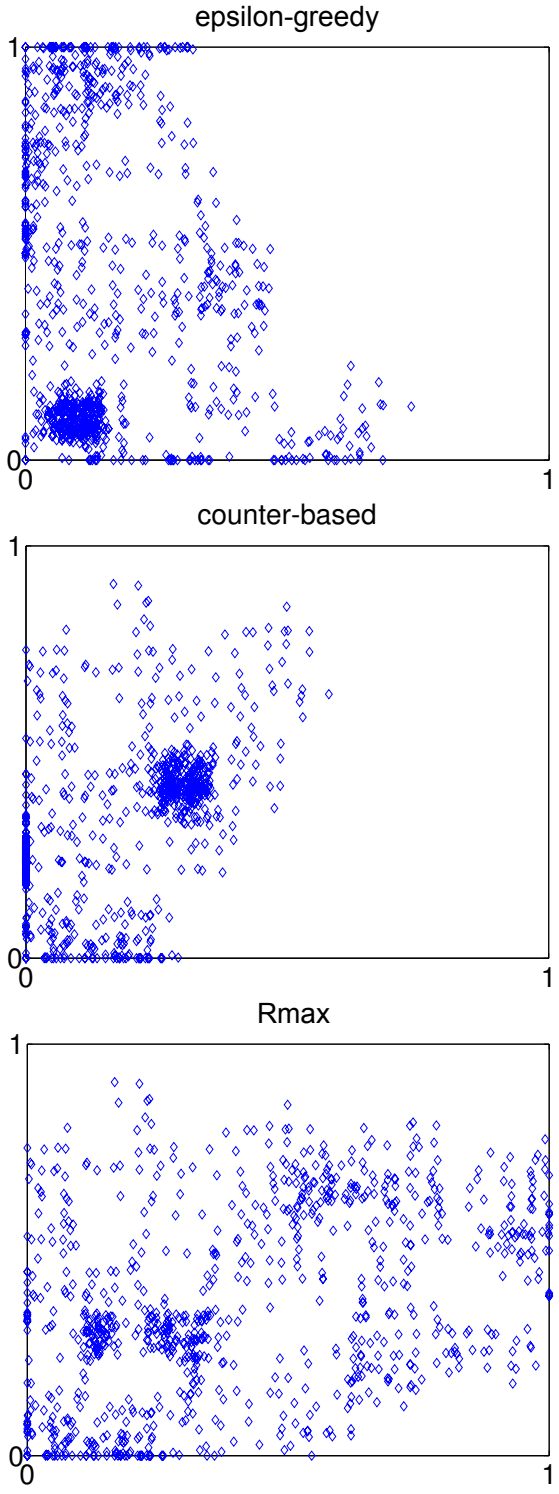


Figure 8.4: State visitation of the first three episodes of a typical run in Express-World. The goal state was not reached with any exploration rules within the first three episodes.

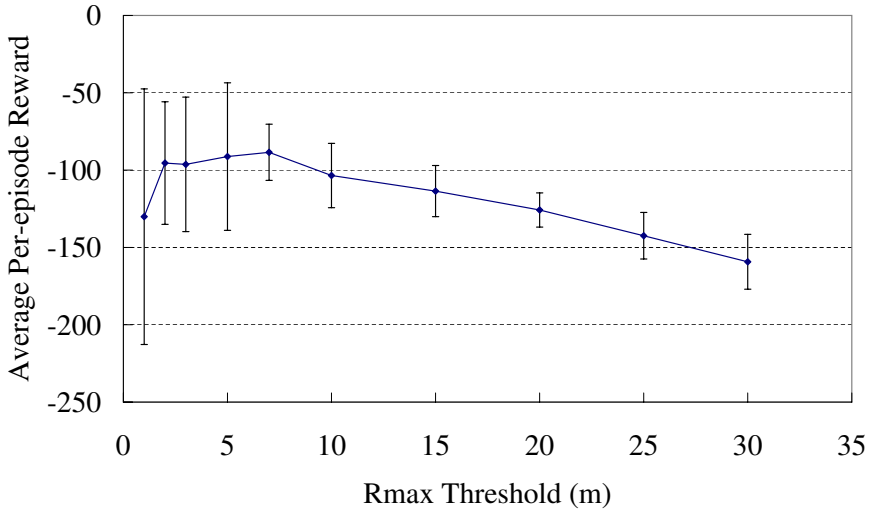


Figure 8.5: Effect of exploration threshold m on average per-episode reward in ExpressWorld.

policies, which explains the large variance in the average per-episode reward. When m gets larger, the agent becomes more conservative and tends to explore more before exploiting. Consequently, learning is more robust and the variance is small. However, being conservative comes with costs as the algorithm delays exploitation while exploring. In between, medium m values work best. Similar patterns are observed in other problems.

8.4 REKWIRE

In this section, we explore a simpler setting of reinforcement learning than what the other parts of the dissertation focus on. In particular, we consider finite-horizon MDPs with a fixed distribution of start states [Fiechter 1994]. This type of MDP is useful for episodic problems such as ExpressWorld in the previous section as well as realistic problems like navigation (*c.f.*, §7.3.2), *etc.* Meanwhile, exploration in these MDPs appears easier than in other MDPs where the agent follows a continuous trajectory without termination (which is the focus of most of the dissertation).

In the following, we first give notation for finite-horizon MDPs, which has a number of similarities to that for infinite-horizon MDPs in §2. Then, we present a a model-free

algorithm with linear value-function approximation. By relating the algorithm to the problem of KWIK linear regression, we can perform a formal analysis for its sample complexity of exploration. Finally, proof-of-concept experiments are discussed.

8.4.1 Preliminaries

In this section, environments are modeled by finite-horizon MDPs which can be described by a six-tuple, $M = \langle \mathcal{S}, \mathcal{A}, T, R, H, \mu_0 \rangle$, where: $\mathcal{S}$, $\mathcal{A}$, T, and R are the state space, action space, transition function, and reward function, similar to the infinite-horizon MDP defined in §2.1; $H \in \mathbb{N}$ is the *horizon*; and $\mu_0 \in \mathcal{P}_{\mathcal{S}}$ is a *start-state distribution*. In general, an H-horizon MDP may have transition probabilities and reward function dependent on the stage. We choose a simpler definition for ease of exposition. The results and analysis in this section may be extended to the general case with minor modifications.

An *episode* is a sequence of H state transitions: $\langle s_1, a_1, r_1, s_2, \cdots, s_H, a_H, r_H, s_{H+1} \rangle$, where $s_1 \sim \mu_0$, s_{H+1} is a terminal state, and rewards and next states are random variables according to the transition and reward functions of the MDP. An agent repeatedly chooses actions until the current episode terminates, and then a new episode starts over again from a start state picked randomly according to μ_0. For convenience, define the set of stages by $[H] = \{1, 2, \cdots, H\}$.

A deterministic, stationary policy maps states and stages to actions: $\pi \in \mathcal{A}^{\mathcal{S} \times [H]}$. Specifically, $\pi(s, h) \in \mathcal{A}$ is the action the agent will take if s is the current state at stage h. Given a policy π, we define the state-value function, $V_h^\pi(s)$, as the expected cumulative reward received by executing π starting from state s at stage h until the episode terminates at stage H. Similarly, the state–action value function, $Q_h^\pi(s, a)$, is the expected cumulative reward received by taking action a in state s at stage h and following π until the episode terminates at stage H. A reinforcement-learning agent attempts to *learn* an optimal policy π^* whose value functions at stage h are denoted by $V_h^*(s)$ and $Q_h^*(s, a)$, respectively. It is known that $V_h^* = \max_\pi V_h^\pi$ and $Q_h^* = \max_\pi Q_h^\pi$. A greedy policy at stage h, denoted π_{Q_h}, with respect to a value function Q_h is one that selects actions with maximum Q-values; namely, $\pi_{Q_h}(s, h) = \arg\max_a Q_h(s, a)$.

The greedy policy with respect to Q_h^* is optimal for stage h. As in infinite-horizon, discounted MDPs, the *Bellman equation* plays a central role to many RL algorithms including the one we will describe: for any $s \in \mathcal{S}, a \in \mathcal{A}, h \in [H]$,

$$Q_h^*(s,a) \quad = \quad R(s,a) + \sum_{s' \in \mathcal{S}} \left(T(s' \mid s,a) \max_{a' \in \mathcal{A}} Q_{h+1}^*(s',a') \right) \tag{8.5}$$

where Q_{H+1}^* is understood to be the zero function.

Given the complete model of a finite MDP (*i.e.*, the six-tuple), standard algorithms exist for finding the optimal value function and the optimal policies [Puterman 1994]; these methods are straightforward modifications to many algorithms surveyed in §3. If the transition and/or reward functions are unknown, the agent has to learn the optimal value function or policy by interacting with the environment.

8.4.2 KWIK Online Linear Regression

In this subsection, we propose an algorithm, REKWIRE (REinforcement learning based on KWIk online REgression), for H-horizon reinforcement-learning problems in which a linear value function is used. The key idea of the algorithm is to reduce the RL problem into a sequence of H instances of KWIK online linear regression problems. A KWIK online linear regression problem is the same as the noisy linear regression problem defined in §5.3.5, except that we do not require the target function to be exactly linear in the input vectors, but that the target function is "almost" linear in the input vectors. This setting is what we call the *semi-linearity assumption* (Assumption 4). As in §5.3.5, we define $\mathcal{X} \subseteq \mathbb{R}^k$ and $\mathcal{Y} = \mathcal{Z} = \mathbb{R}$, where k is the dimension of inputs.

Assumption 4 *We make the following assumptions for the KWIK online linear regression problem:*

A. (Bounded-input assumption) $\|\mathbf{x}_t\|_2 \leq 1$ *for all* t.

B. (Semi-linearity assumption) There exist some (unknown) vector $\mathbf{w}^* \in \mathbb{R}^k$ *and a small number* $\xi \in [0,1)$ *such that* $\|\mathbf{w}^*\|_2 \leq 1$ *and* $|\mathbf{E}[z_t|\mathbf{x}_t] - \mathbf{w}^* \cdot \mathbf{x}_t| \leq \xi$ *for all* t. *We call the quantity* ξ *the* slack *value.*

Another assumption is needed for the foundation of our algorithm.

Assumption 5 *Under certain conditions $\mathcal{C}$ on the process generating the samples $[\langle \mathbf{x}_t, y_t \rangle]_{t\in\mathbb{N}}$, there exists a KWIK online linear regression algorithm $\mathbf{A}_0$ with a KWIK bound $B_0(k, \frac{1}{\epsilon}, \frac{1}{\delta})$ and a per-step computation complexity $\tau_0(k, \frac{1}{\epsilon}, \frac{1}{\delta})$.*

Before ending this section, we give an example to shed some light on what $\mathcal{C}$ might look like. Some notation is in order. Let $\mathbf{z}_t = [y_1, y_2, \cdots, y_t]^\top$ and $X_t \in \mathbb{R}^{t\times k}$ be the design matrix up to time t: $X_t = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t]^\top$. Then, $\bar{y}_{t+1} = \mathbf{a}_{t+1}^\top \mathbf{z}_t$ gives the *least-squares* prediction of y_{t+1}, where $\mathbf{a}_{t+1} = X_t(X_t^\top X_t)^\dagger \mathbf{x}_{t+1}$, $M^\top$ and $M^\dagger$ denotes the transpose and pseudo-inverse of matrix M, respectively. Define $\mathbf{e}_t \stackrel{\text{def}}{=} [\epsilon_1^*, \epsilon_2^*, \cdots, \epsilon_t^*]^\top$. Since $|\epsilon_i^*| \le \xi$ for all i (Assumption 4B), we have $\left|\mathbf{a}_t^\top \mathbf{e}_t\right| \le \xi \left\|\mathbf{a}_t\right\|_1$. This inequality is tight when the signs of components of $\mathbf{a}_t$ and $\mathbf{e}_t$ coincide (or are completely opposite). It turns out that if the error ϵ_t^* "mixes quickly" with $\mathbf{a}_t$ over time, then we have admissible KWIK online linear regression algorithms:

Theorem 27 *If $\left|\mathbf{a}_t^\top \mathbf{e}_t\right| = \mathcal{O}\left(t^{-\alpha}\xi\right)$ as $t \to \infty$ for some constant $\alpha > 0$, then noisy linear-regression (Algorithm 16) is a KWIK algorithm under Assumption 4.*

We conjecture that this sufficient condition is reasonable in some natural problems of practical interest, and thus will use noisy linear-regression as $\mathbf{A}_0$ in the experiments.

8.4.3 Algorithm

We assume a set of d features are predefined: $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \to [-1, 1]^k$. A Q-function can then be represented compactly by a weight vector $\mathbf{w}_h \in \mathbb{R}^k$ for each $h \in [H]$: $\hat{Q}_h(s, a) = \mathbf{w}_h \cdot \boldsymbol{\phi}(s, a)$. We make a semi-linearity assumption for the value function at every stage. Remember that outputs in the KWIK online regression are in $[-1, 1]$; we need to re-scale the value function to the same range by dividing Q_h^* by H:

Assumption 6 *(Semi-linearity assumption of the optimal value function) For every stage $h \in [H]$, there exists some (unknown) vector $\mathbf{w}_h^* \in \mathbb{R}^k$ and a small number $\xi_h > 0$ such that $\left\|\mathbf{w}_h^*\right\|_2 \le 1$ and*

$$\left| \frac{Q_h^*(s, a)}{H} - \mathbf{w}_h^* \cdot \boldsymbol{\phi}(s, a) \right| \quad \le \quad \xi_h \tag{8.6}$$

for all s and a. Whether it is required to know ξ_h depends on whether such information is needed by $\mathbf{A}_0$.

Algorithm 8.4.3 gives a formal description of REKWIRE. Basically, the algorithm learns the optimal value functions Q_h^* by treating them as H related KWIK online linear regression problems. It runs H copies of the base algorithm $\mathbf{A}_0$ to update the weight vector $\mathbf{w}_h$ for stage h. By the Equation 8.5, the Q-function at stage h is defined recursively as the sum of immediate reward at stage h and the expected optimal Q-value of the next states. Therefore, the algorithm improves its value-function estimates by performing Bellman-backup-style updates. A central idea behind the efficiency of the reduction is that we only use a backup value when it is "known". A backup value is "known" when the prediction made by $\mathbf{A}_0$ is valid, and thus by Assumption 5 must be near-accurate. Figure 8.6 gives a simple H-horizon example for $H = 3$ and illustrates how REKWIRE chooses actions and computes backup values.

A quick observation about REKWIRE is that, if the per-step computation complexity of $\mathbf{A}_0$ is τ_0, then the per-step computation complexity of REKWIRE is $\mathcal{O}(|\mathcal{A}|\,\tau_0)$, when execution of Lines 17–21 are amortized to every timestep. So, the per-step computation complexity scales nicely from regression problems to sequential decision making in MDPs, in contrast to algorithms such as state-space discretization [Chow and Tsitsiklis 1989] that scales exponentially in the dimension of $\mathcal{S}$ and sparse sampling [Kearns et al. 2002] that scales exponentially in the horizon H. We next turn to the more difficult questions of bounding the sample complexity of exploration and value-function approximation error.

8.4.4 Analysis

The first main result in this section is the following theorem.

Theorem 28 *Suppose Assumption 6 holds. If $\mathbf{A}_0^{(h)}$ is run with parameters ϵ_h and δ_h in REKWIRE, then:*

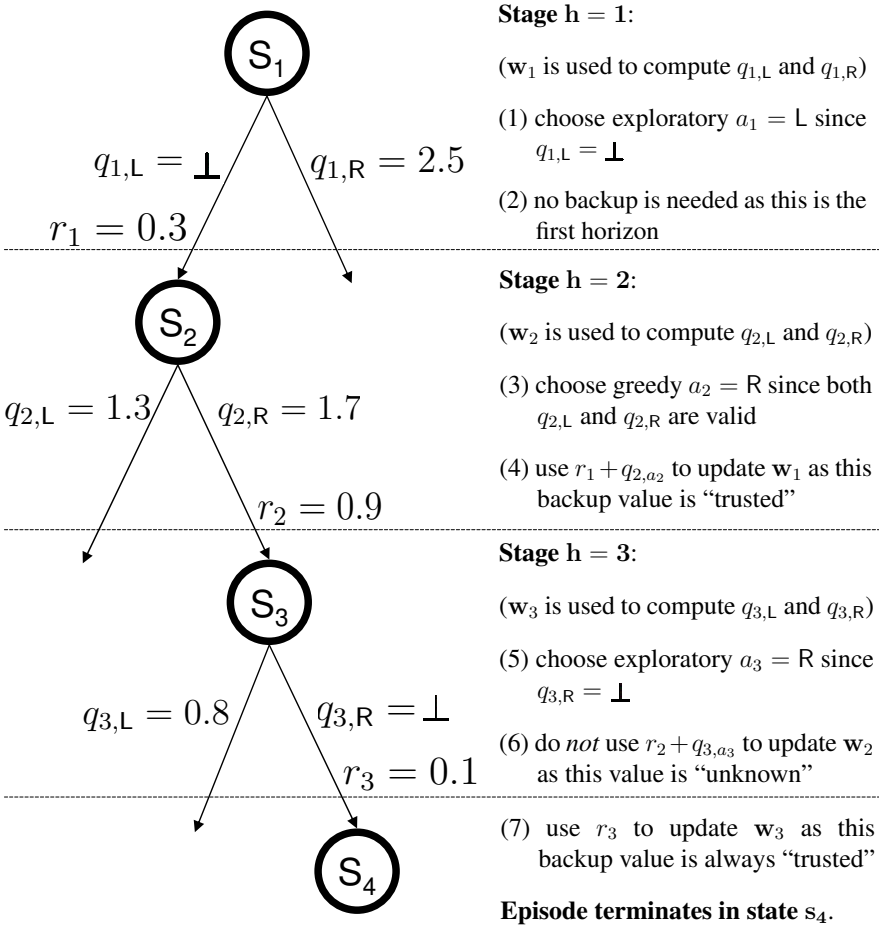


Figure 8.6: An illustration of the operations of REKWIRE in a 3-horizon MDP. Two actions are allowed in every state: $\{\mathsf{L}, \mathsf{R}\}$. The same notation as in Algorithm 8.4.3 is used.

I. The number of $\perp$s outputted in stage $h \in [H]$ is at most

$$\sum_{l=h}^{H} B_0\left(k, \frac{1}{\epsilon_l}, \frac{1}{\delta_l}\right);$$

II. The total number of $\perp$s outputted during the whole run of REKWIRE in all stages is at most

$$\sum_{h=1}^{H} \left(hB_0\left(k, \frac{1}{\epsilon_h}, \frac{1}{\delta_h}\right)\right);$$

III. With probability at least $1 - \sum_{l=1}^{H} \delta_l$, all valid Q-value predictions at stage h differ from the true values by at most

$$H \sum_{l=h}^{H} (\epsilon_l + \xi_l).$$

0: Inputs: $\mathcal{S}$, $\mathcal{A}$, H, $\boldsymbol{\phi}$, ϵ_h, and δ_h for $h \in [H]$.
1: Initialize H copies of $\mathbf{A}_0$, one for each $h \in [H]$. The copy at stage h is run with parameters ϵ_h and ξ_h, and is denoted by $\mathbf{A}_0^{(h)}$.
2: **for** episode $i = 1, 2, 3, \cdots$ **do**
3: **for** stage $h = 1, 2, 3, \cdots, H$ **do**
4: Observe state s_h.
5: **for all** $a \in \mathcal{A}$ **do**
6: Use $\mathbf{A}_0^{(h)}$ to compute $q_{h,a} \in [0, H] \cup \{\perp\}$ as a prediction for $Q_h^*(s_h, a)$. Here, if $\mathbf{A}_0^{(h)}$ gives a valid prediction, then this prediction has to be multiplied by H to obtain $q_{h,a}$ due to the normalization (Equation 8.6) we have used.
7: **end for**
8: **if** $q_{h,a} = \perp$ for some $a \in \mathcal{A}$ **then**
9: $a_h \leftarrow a$ {do exploration}
10: $L_h \leftarrow$ FALSE $\{Q_h^*(s_h, a_h)$ is "unknown"$\}$
11: **else**
12: $a_h \leftarrow \arg\max_a q_{h,a}$ {do exploitation}
13: $L_h \leftarrow$ TRUE $\{Q_h^*(s_h, a_h)$ is "known" and $q_{h,a}$ is "trusted"$\}$
14: **end if**
15: Take action a_h and observe reward r_h.
16: **end for**
17: **for** $h = 2, 3, \cdots, H$ **do**
18: **if** $L_h =$ TRUE **then**
19: Use $\left(\boldsymbol{\phi}(s_{h-1}, a_{h-1}), \frac{r_{h-1} + q_{h,a_h}}{H}\right)$ as an example for $\mathbf{A}_0^{(h-1)}$ to update $\mathbf{w}_{h-1}$.
20: **end if**
21: **end for**
22: Use $\left(\boldsymbol{\phi}(s_H, a_H), \frac{r_H}{H}\right)$ as an example for $\mathbf{A}_0^{(H)}$ to update $\mathbf{w}_H$. {terminating rewards are always "trusted"}
23: **end for**

The following corollary, which follows immediately from Theorem 28, indicates that the KWIK bound and error bound of the KWIK online linear regression algorithm $\mathbf{A}_0$ scale nicely to the analogous quantities in the more complicated, H-horizon RL problem.

Corollary 2 *If $\epsilon_h = \epsilon_0$, $\delta_h = \delta_0$, and $\xi_h = \xi_0$ for all $h \in [H]$ in Theorem 28, then:*

I. The number of $\perp$s outputted at stage h is

$$\mathcal{O}\left(HB_0\left(k, \frac{1}{\epsilon_0}, \frac{1}{\delta_0}\right)\right);$$

II. The total number of $\perp$s outputted during a whole run of REKWIRE *in all stages is*

$$\mathcal{O}\left(H^2 B_0\left(k, \frac{1}{\epsilon_0}, \frac{1}{\delta_0}\right)\right);$$

III. With probability at least $1 - H\delta_0$*, all valid Q-value predictions at stage* h *differ from the true values by at most*

$$H(H-h+1)\left(\epsilon_0+\xi_0\right) = \mathcal{O}\left(H^2\left(\epsilon_0+\xi_0\right)\right).$$

Using Corollary 2, we can prove the following theorem about the sample complexity of exploration of REKWIRE. Our focus is to provide the first polynomial sample complexity bound, although improved bounds are possible with a more careful analysis.

Theorem 29 *Given any* $\epsilon, \delta > 0$ *and* k *features that satisfy Assumption 6 with* $\xi_h = \mathcal{O}\left(\frac{\epsilon}{H^3}\right)$*, if we run* $\mathbf{A}_0^{(h)}$ *with* $\epsilon_h = \mathcal{O}\left(\frac{\epsilon}{H^3}\right)$ *and* $\delta_h = \mathcal{O}\left(\frac{\delta}{H}\right)$ *in* REKWIRE*, then the policy used by the agent is* ϵ*-optimal except in*

$$\mathcal{O}\left(\frac{H^3}{\epsilon}\cdot B_0\left(k, \frac{H^3}{\epsilon}, \frac{H}{\delta}\right)\cdot \ln\frac{1}{\delta}\right)$$

episodes, with probability at least $1-\delta$*.*

8.4.5 An Extension to the Discounted Case

While we have focused on finite-horizon RL problems in this section, it is sometimes easier to model environments by infinite-horizon MDPs with a discount factor, as is defined in §2.1. Changes in notation and terminology are necessary since there is no notion of horizon in this setting. Specifically, we only need to consider policies that maps states to actions: $\pi \in \mathcal{A}^{\mathcal{S}}$. The value functions, such as $Q^\pi(s,a)$ and $Q^*(s,a)$, are defined as the expected γ-discounted cumulative reward. These are what we have focused on in the rest of the dissertation.

In light of Lemma 2, we may convert a γ-discounted, infinite-horizon MDP M_γ into an H-horizon MDP M_H so that the optimal value functions of M_H and M_γ differ by at most ϵ, provided

$$H > \frac{1}{1-\gamma}\ln\frac{1}{\epsilon(1-\gamma)}.$$

It is worth mentioning that even if the optimal value function Q^*_γ of M_γ is semi-linear, the value functions in M_H, Q^*_h, need not be semi-linear. We note that this problem may be resolved by using different sets of features at different stages. That

is, we require features $\boldsymbol{\phi}_h : \mathcal{S} \times \mathcal{A} \to [-1, 1]^d$ at stage h, and assume that Q_h^* satisfies Assumption 6 with small slack ξ_h. REKWIRE can then be applied.

8.4.6 Related Work

Our work is most relevant to the original KWIK online linear regression framework proposed by Strehl and Littman [2008b]. The only difference in problem formulation is that we make a semi-linearity assumption while they assume exact linearity. This change is necessary if we allow Bellman-backup-style updates on the value functions since the backup value (*i.e.*, $\frac{r_{h-1}+q_{h,a_h}}{H}$ in Line 19 of Algorithm 8.4.3) is unavoidably biased and it is unreasonable to assume the target function at stage $h-1$ remains linear for all possible biases introduced in stage h.

The second significant difference is how KWIK online linear regression is applied to RL problems. Strehl and Littman [2008b] adopt a *model-based* approach: they assume the MDP state transitions are governed by a set of linear equations with Gaussian white noise and apply KWIK online linear regression to learn the transition matrices, finally solving the learned MDP model to obtain a policy that either explores or exploits. Even if an MDP can be accurately modeled as a linear system, solving a continuous MDP remains a challenging task [Chow and Tsitsiklis 1989; Kearns et al. 2002]. A similar problem arises in the metric E^3 algorithm [Kakade et al. 2003]. In contrast, the *model-free* approach taken in this paper avoids this problem completely by learning the value function directly. With a learned linear value function, finding the greedy action takes only $\mathcal{O}(|A|\,\tau_0)$ time per step.

The KWIK online linear regression framework we described is related to the online learning model of linear functions (*e.g.*, Cesa-Bianchi et al. [1996]). In this model, like ours, input data are not assumed to be *i.i.d.* Cumulative absolute and squared prediction error bounds are developed under various assumptions. The main difference between that model and ours is that we require the learner to be aware of its prediction accuracy.

Our algorithm shares some similarity with the grow-support algorithm by Boyan and Moore [1995] in that we both use value-function estimates for backups when the

estimates are "trusted". But grow-support assumes complete knowledge about the MDP model and thus focuses on stability issues when combining function approximation and dynamic programming. In contrast, REKWIRE is a learning algorithm and has to explore in the MDP to gather information needed for computing the optimal policy.

Finally, Peters and Schaal [2007] proposed an interesting reduction from RL to reward-weighted regression. While they consider the specific task of following a given trajectory in rigid-body systems whose dynamics are governed by a set of equations with unknown parameters, this paper focuses on learning in general MDPs where the learner is not provided with such target trajectories.

8.4.7 Experiments

We demonstrate REKWIRE on a few benchmark problems as a proof of concept. Noisy linear-regression was used as the base algorithm $\mathbf{A}_0$. Sarsa(0) with ϵ-greedy and Boltzmann exploration rules were used for comparison.

Setup

The same three finite-horizon problems in Figure 8.2 were used. In ContCombLock, the *left* action always resets the agent to the start state s_1, and the *right* action takes the agent from state s to a new state $s' = s + 0.05 + \Delta$ where $\Delta \sim \mathcal{N}(0, 0.0125)$ is Gaussian noise. If the agent reaches a state $s > 0.95$, the episode terminates. Every step results in a -1 reward, as before. Therefore, the optimal policy is to always choose *right*, and on average each episode takes about 19 to 20 steps to finish. We set $H = 25$. To separate the issue of exploration from that of feature selection, we used $\phi_h(s, \text{LEFT}) = Q_h(s, \text{LEFT}) \cdot [1, 0, n_1, n_2]^\top$ and $\phi_h(s, \text{RIGHT}) = Q_h(s, \text{RIGHT}) \cdot [0, 10, n_1, n_2]^\top$, where n_1 and n_2 were noisy values uniformly distributed in $[-0.5, 0.5]$, and $Q_h(s, a) \approx Q_h^*(s, a)$ was computed by the Bellman equation (Equation 8.5) on the discretized MDP. Clearly, $w_h = [1, 0.1, 0, 0]^\top$ yields a quite accurate linear approximation to Q_h^*, and the noise was added to make the problem more interesting.

In MountainCar, the start state is $s_1 = [0, 0]$ and we set $H = 75$. As before, $\phi_h(s, a) = Q_h(s, a) \cdot [1, n_1, n_2]^\top$, and hence $w_h = [1, 0, 0]^\top$ yields a quite accurate linear

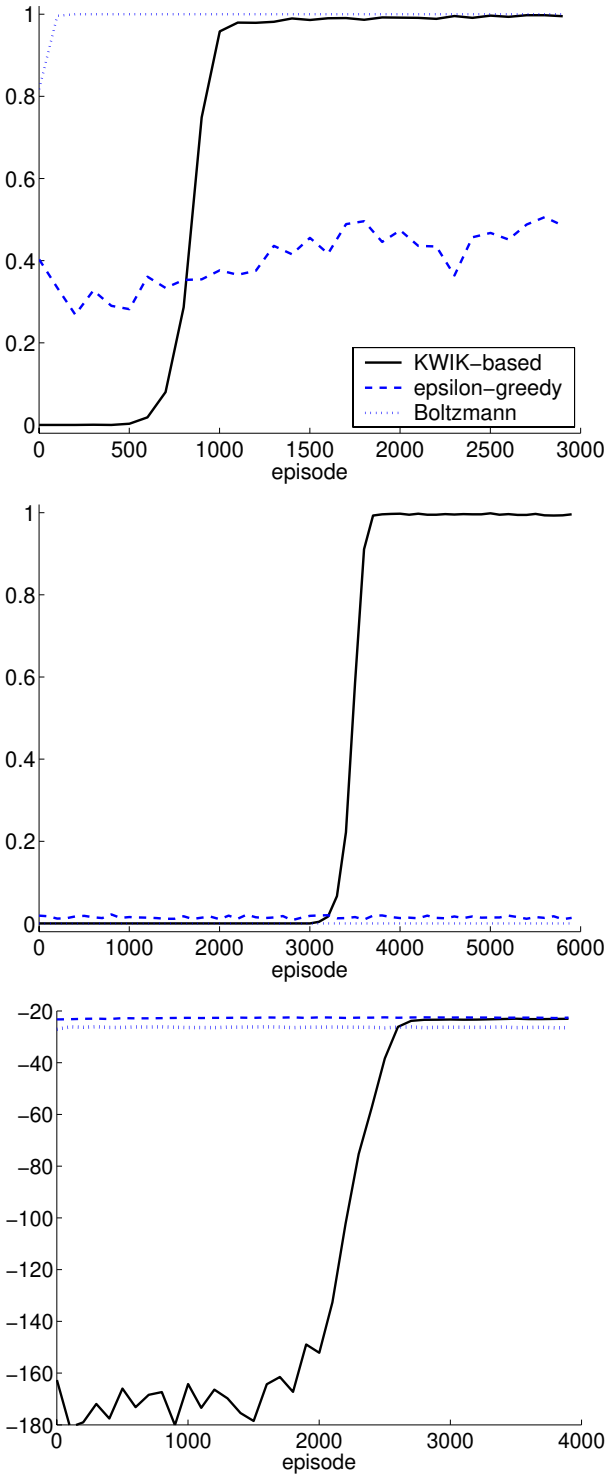


Figure 8.7: Probabilities of reaching the goal states in ContCombLock (top), MountainCar (middle), and per-episode rewards in PuddleWorld (bottom). All results are averaged over 20 runs.

approximation to Q_h^*.

In ExpressWorld, we set $H = 45$. As before, $\phi_h(s,a) = \hat{Q}_h(s,a) \cdot [1, n_1, n_2]^\top$, and hence $w_h = [1,0,0]^\top$ yields a quite accurate linear approximation to Q_h^*.

Results

Figure 8.7 summarizes the results on all three problems. We picked parameters that seemed to work best for each algorithm. Since the number of steps to the goal states is very close to H in ContCombLock and MountainCar, we instead plotted the probabilities of reaching the goal within H steps. In contrast, we evaluated per-episode reward in ExpressWorld as it is necessary to distinguish whether the agent discovers the express lane or not.

It is observed that our algorithm consistently solved all three problems, while ϵ-greedy and Boltzmann rules worked satisfactorily for some and failed for others. For example, ϵ-greedy was inefficient in ContCombLock, as expected; the Boltzmann rule failed to converge to the optimal policy in ExpressWorld although it quickly learned a good policy early on. Our algorithm, however, seemed too conservative and converged more slowly. A partial explanation is that it learned H weight vectors, while Sarsa learned only one weight vector and used it to compute a policy in all stages. It remains an interesting open question how we may avoid representing the value function using H weights in REKWIRE, which is particularly important when it is applied to discounted problems.

We next take a closer look at REKWIRE in ContCombLock, whose value functions can be easily visualized. To make the problem more interesting, we ran REKWIRE with RBF features with 4 RBF centers located evenly in the state space $[0,1]$. Figure 8.8 plots the probability of reaching the goal within $H = 25$ steps for our algorithm, which is consistently high after around $25,000$ episodes.[6] In contrast, both ϵ-greedy and Boltzmann rules failed this task in our experiments.

[6]Since the feature vector has 10 dimensions (1 constant feature plus 4 RBF centers per action) and $H = 25$, each parameter required about 1000 data to estimate.

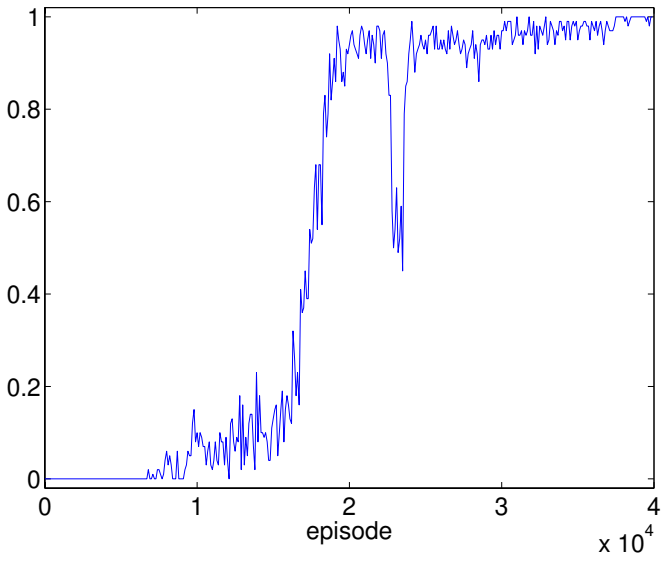


Figure 8.8: Probability of reaching the goal within $H = 25$ steps in ContCombLock when RBF features were used. The results were computed based on a typical run and the probabilities were computed using every other block of 100 episodes.

Figure 8.9 gives a few snapshots of the value function $\hat{V}_h(s) = \max_a \hat{Q}_h(s,a)$ computed by REKWIRE: $\hat{V}_h(s,a)$ is a valid prediction only when $\hat{Q}_h(s,a)$ is valid for both actions; otherwise, $\hat{V}_h(s,a) = \perp$ and the value is not plotted in the figure. At the beginning of learning, most states' values are unknown. The algorithm has to work backwards by learning Q^*_h from larger h to smaller h, as shown in the plots. The plots show that the "known" region essentially stopped growing after about 25000 episodes, and the policy stablized after that, which is consistent with the learning curve in Figure 8.8. Comparing to the true value function, we found that the value function computed by REKWIRE was indeed very close to the true value for "known" regions, exactly as we would expect.

8.5 Proofs

This section provides detailed proofs of all technical lemmas used in this chapter.

8.5.1 Proofs of Lemmas for Theorem 22

PROOF (of Lemma 38). The proof is by mathematical induction. For $t = 1$ and any (s, a), $Q_1(s, a) = V_{\max} \geq Q^*$ due to Line 2 of Algorithm 27, and thus $Q_1(s, a) \geq Q^*(s, a) - \epsilon_1/(1-\gamma)$. Assume $Q_t(s, a) \geq Q^*(s, a) - \epsilon_1/(1-\gamma)$ for some t and all (s, a), and a successful update occurs in timestep t on (s_t, a_t). Then,

$$Q_{t+1}(s_t, a_t) \;\geq\; \mathfrak{B}Q_t(s_t, a_t) - \epsilon_1 \;\geq\; \mathfrak{B}Q^*(s_t, a_t) - \frac{\gamma\epsilon_1}{1-\gamma} - \epsilon_1 \;=\; Q^*(s_t, a_t) - \frac{\epsilon_1}{1-\gamma}$$

where the first step is due to the assumption that $\hat{\mathfrak{B}}$ is an ϵ_1-approximation of $\mathfrak{B}$, the second due to Lemma 4 and the inductive hypothesis, and the last due to the fixed-point property of Q^*. Also observe that $Q_{t+1}(s, a) = Q_t(s, a)$ for all $(s, a) \neq (s_t, a_t)$, and thus the optimism condition continues to hold for these unaffected state–action pairs. □

8.5.2 Proofs of Lemmas for Theorem 23

PROOF (of Lemma 39). Note that each time a successful update occurs for (s_t, a_t) at timestep t, $Q_t(s_t, a_t)$ decreases by at least ϵ_1. Since $Q_1(s, a) = V_{\max}$ and $Q_t(s, a) \geq 0$ for all t, at most $V_{\max}/\epsilon_1$ successful updates can happen. Hence, the total number of successful updates during the whole run of randomized RTDP is bounded by $\kappa_s = |S|\,|A|\,V_{\max}/\epsilon_1$. Now, note that when a state-action pair, (s, a), is first experienced, it will always perform an attempted update. Next, another attempted update of (s, a) will occur only when at least the Q-value of some state–action pair has been successfully updated (due to Line 8 of Algorithm 27) since the last attempted update of (s, a). Thus, the total number of attempted updates is bounded by κ_a. □

PROOF (of Lemma 40). Let t be an arbitrary timestep when an attempted update occurs on $Q_t(s_t, a_t)$, and let $\{(r_{t1}, s'_{t1}), (r_{t2}, s'_{t2}), \ldots, (r_{tm}, s'_{tm})\}$ be the m sampled transitions. Define m random variables as follows: for $i = 1, 2, \ldots, m$,

$$x_i \stackrel{\text{def}}{=} r_{ti} + \gamma \max_{a \in \mathcal{A}} Q_t(s'_{ti}, a).$$

Since $r_{ti} \in [0, 1]$ and $Q_t(s, a) \in [0, V_{\max}]$, we have $x_i \in [0, \gamma V_{\max} + 1]$. Furthermore, $\mathbf{E}[x_i] = \mathfrak{B}Q_t(s_t, a_t)$ by definition. It follows immediately from Lemma 52 that for any

$\delta_1 \in (0,1)$, the following holds with probability at least $1-\delta_1$: $|q - \mathfrak{B}Q_t(s_t,a_t)| \leq \epsilon_1$, if

$$m = \frac{(\gamma V_{\max}+1)^2}{2\epsilon_1^2} \ln \frac{2}{\delta_1}$$

Since there are at most κ_a attempted updates, due to Lemma 39, the lemma is proved by setting $\delta_1 = \frac{\delta}{2\kappa_a}$ and a union bound. □

Proof (of Lemma 41). According to Lemma 40, with probability at least $1-\delta/2$, Equation 8.1 holds for all timesteps t when an attempted update occurs. Therefore, with probability at least $1-\delta/2$, in an entire run of randomized RTDP, if $(s_t,a_t) \notin K_t \stackrel{\text{def}}{=} K_{Q_t}(\epsilon(1-\gamma)/4)$, the attempted update will be successful because

$$q - Q_t(s_t,a_t) \;<\; \mathfrak{B}Q_t(s_t,a_t) - Q_t(s_t,a_t) + \epsilon_1 \;\leq\; -\frac{\epsilon(1-\gamma)}{4} + \epsilon_1 \;=\; -\epsilon_1.$$

We have used the fact that $\epsilon_1 = \epsilon(1-\gamma)/8$. □

Proof (of Lemma 42). Equation 8.1 guarantees that every sampling-based approximate Bellman backup performed in Line 9 is an ϵ_1-approximation of the true Bellman backup, and thus the proof parallels that of Lemma 38. □

8.5.3 Proofs of Lemmas for Theorem 24

Proof (of Lemma 43). Consider a fixed state–action pair (s,a). Its associated action–value estimate is initialized to $V_{\max}$ and cannot be negative during the whole run of the algorithm. Each time $Q(s,a)$ is successfully updated, it decreases by at least ϵ_1. Thus, $Q(s,a)$ cannot be updated more than $V_{\max}/\epsilon_1$ times, and thus at most κ_s successful updates are possible.

Consider a fixed state–action pair (s,a). Once (s,a) is experienced for the m-th time, an attempted update will occur. Suppose that an attempted update of (s,a) occurs during timestep t. Afterward, for another attempted update to occur during some later timestep t', it must be the case that a successful update of some state–action pair (not necessarily (s,a)) has occurred on or after timestep t and before timestep t'. Therefore, there are at most $1+\kappa_s$ attempted updates of (s,a), and thus κ_a attempted updates are possible. □

Proof (of Lemma 44). Fix any timestep k_1 (and the complete history of the agent up to k_1) such that $(s_{k_1},a_{k_1}) \notin K_{k_1}$, $L(s_{k_1},a_{k_1}) = \text{TRUE}$, and $C(s_{k_1},a_{k_1}) = 0$. In

other words, if (s_{k_1}, a_{k_1}) is experienced $m-1$ more times after timestep k_1, then an attempted update will occur.

Let $\mathcal{D} = \{(s[1], r[1]), \ldots, (s[m], r[m])\} \in (\mathcal{S} \times \mathbb{R})^m$ be any sequence of m independent next state and immediate reward tuples for (s_{k_1}, a_{k_1}). Due to the Markov assumption, whenever the agent is in state s_{k_1} and chooses action a_{k_1}, the resulting next state and immediate reward are chosen independently of the history of the agent. Thus, the probability that (s_{k_1}, a_{k_1}) is experienced $m-1$ more times *and* that the resulting next state and immediate reward sequence equals $\mathcal{D}$ is at most the probability that $\mathcal{D}$ is obtained by m independent draws from the transition and reward distributions for (s_{k_1}, a_{k_1}). Therefore, it suffices to prove this lemma by showing that the probability that a random sequence $\mathcal{D}$ could cause an unsuccessful update of (s_{k_1}, a_{k_1}) is at most $\delta/3$. We prove this statement next.

Given the random sequence in $\mathcal{D}$, we may formulate the problem of approximating a Bellman backup while maintaining an optimistic value function as a subinterval prediction problem studied in §5.3.4. Specifically, the i-th output/interval is $[\mathfrak{B}Q_{k_i}(s_t, a_t), \mathfrak{B}Q_{k_1}(s_t, a_t)]$, the i-th observation is $r[i] + \gamma V_{k_i}(s[i])$ whose expectation is $\mathfrak{B}Q_{k_i}(s_t, a_t)$. If we run coin-learning and make the first valid prediction after observing m samples, then Theorem 12 guarantees that the new Q-value is still optimistic with probability at least $1 - \frac{\delta}{3\kappa_a}$ when m is set by Equation 8.2.

Finally, using a union bound over all possible timesteps k_1 satisfying the condition above, which is at most κ_a, we complete the proof. □

Proof (of Lemma 45). It can be shown, by a similar argument as in the proof of Lemma 44, that at all timesteps t where an attempted update occurs, we have with probability at least $1 - \delta/3$ that $q - \mathfrak{B}Q_t(s_t, a_t) > -\epsilon_1$, where $q = U(s_t, a_t)/m$ is the attempted update value. Now assume the inequality above is true for all attempted updates, and then the proof proceeds by mathematical induction in a similar manner as in the proof of Lemma 42. □

Proof (of Lemma 46). The proof is by contradiction. Suppose an unsuccessful update occurs at timestep t, $L_{t+1}(s_t, a_t) = \text{FALSE}$, and $(s_t, a_t) \notin K_{t+1}$. Let $k_1 < k_2 < \cdots < k_m = t$ be the most recent m timesteps in which (s_t, a_t) is experienced. Because of

event A_{DQL}, we have $(s_t, a_t) \in K_{k_1}$. Since the update is unsuccessful on timestep t, we have that $K_t = K_{t+1}$, which in turn implies $(s, a) \notin K_t$. Therefore, there must exist some timestep t' (so that $t > t' > k_1$) in which a successful update occurs. Thus, $L_{t+1}(s_t, a_t) = \text{TRUE}$ since it is set so at timestep t' in Line 12 of Algorithm 28, which contradicts our assumption. □

Proof (of Lemma 47). First, observe that $Q(s, a)$ cannot be updated more than $V_{\max}/\epsilon_1$ times since every update will decrease the value by at least ϵ_1, and the value is non-negative.

Suppose at some timestep t, $(s_t, a_t) \notin K_t$ and $L(s_t, a_t) = \text{FALSE}$ (implying the last attempted update was unsuccessful). By Lemma 46, we have that $(s_t, a_t) \in K_{t'+1}$ where t' was the time of the last attempted update of (s, a). Thus, some successful update has occurred since timestep $t' + 1$. By the rules of Algorithm 28, we have that $L(s_t, a_t)$ will be set to TRUE and, by event A_{DQL}, the next attempted update will succeed.

Now, suppose at timestep t, $(s_t, a_t) \notin K_t$ and $l(s_t, a_t) = \text{TRUE}$. Within at most m more experiences of (s_t, a_t), an attempted update of $Q(s_t, a_t)$ will occur. Suppose this attempted update takes place at time $t'' > t$ and that the m most recent experiences of (s_t, a_t) happened at times $k_1 < k_2 < \cdots < k_m = t''$. By event A_{DQL}, if $(s_t, a_t) \notin K_{k_1}$, the update will be successful. Otherwise, if $(s_t, a_t) \in K_{k_1}$, then some successful update must have occurred between times k_1 and t (since $K_{k_1} \neq K_t$). Hence, even if the update is unsuccessful, $L(s_t, a_t)$ remains TRUE, $(s_t, a_t) \notin K_{t''+1}$ will hold, and the next attempted update of (s_t, a_t) will be successful.

In either case, if $(s_t, a_t) \notin K_t$, then within at most $2m$ more experiences of (s_t, a_t), a successful update of $Q(s_t, a_t)$ will occur. This observation concludes the proof of the lemma together with the previous observation that every state–action value can be changed at most $V_{\max}/\epsilon_1$ times. □

8.5.4 Proof of Theorem 25

Proof. (of Lemma 48) If we treat decision making in each state as an A-arm bandit problem, finding the optimal action for that state becomes one of finding an ϵ-optimal action in the bandit problem. This bandit problem is the one used by Mannor and

Tsitsiklis [2004] to establish a sample complexity lower bound (Lemma 11) in the PAC model.

By construction of the MDP in Figure 8.1, there is at most one optimal action in each state $i \in [N]$. Thus, if any bandit algorithm $\mathbf{A}'$ can guarantee with probability at least $1 - \delta_i$ that at most m_i sub-optimal actions are taken in a whole run, then we can turn it into a bandit algorithm with a PAC sample complexity of $2m_i + 1$: we simply run $\mathbf{A}'$ for $2m_i + 1$ steps and the majority action must be ϵ-optimal with probability at least $1 - \delta_i$. In other words, the lower bound above for PAC sample complexity results immediately in a lower bound for the total number of sub-optimal actions taken by $\mathbf{A}'$, and thus

$$m_i \geq \frac{c_1 A}{\epsilon^2} \ln \frac{c_2}{\delta_i}$$

for appropriately chosen constants c_1 and c_2. Reorganizing terms gives the desired result. □

8.5.5 Proof of Theorem 27

If $\hat{y}_t$ is valid, the agent does not see y_t, and thus we can assume without loss of generality that $\hat{y}_t$ is unknown for $t = 1, 2, \cdots$ up to some indefinite T. Let $\bar{y}_{t+1} = \mathbf{a}_{t+1}^\top \mathbf{z}_t$ be the least-squares prediction of y_{t+1}, where $\mathbf{a}_{t+1} = X_t \left(X_t^\top X_t\right)^\dagger \mathbf{x}_{t+1}$. Then,

$$\begin{aligned}
|\mathbf{E}[\bar{y}_{t+1}] - y_{t+1}| &= \left|\mathbf{E}[\mathbf{a}_{t+1}^\top \mathbf{z}_t] - y_{t+1}\right| \\
&\leq \left|\mathbf{E}[\mathbf{a}_{t+1}^\top \mathbf{z}_t] - \mathbf{w}_*^\top \mathbf{x}_{t+1}\right| + \left|\mathbf{w}_*^\top \mathbf{x}_{t+1} - y_{t+1}\right| \\
&= \left|\mathbf{E}[\mathbf{a}_{t+1}^\top \mathbf{z}_t] - \mathbf{w}_* X_t^\top \mathbf{a}_{t+1}\right| + \left|\mathbf{w}_*^\top \mathbf{x}_{t+1} - y_{t+1}\right| \\
&\leq \left|\mathbf{E}[\mathbf{a}_{t+1}^\top \mathbf{z}_t] - \mathbf{w}_* X_t^\top \mathbf{a}_{t+1}\right| + \xi \\
&\leq \left|\sum_{\tau=1}^{t} \mathbf{a}_{t+1,\tau} \left(\mathbf{E}[y_\tau] - \mathbf{w}_*^\top \mathbf{x}_t\right)\right| + \xi \\
&= \left|\sum_{\tau=1}^{t} \mathbf{a}_{t+1,\tau}^\top \mathbf{e}_\tau\right| + \xi \\
&= \left|\mathbf{a}_{t+1}^\top \mathbf{e}_t\right| + \xi,
\end{aligned}$$

where the first inequality follows from the triangle inequality, the second inequality from Assumption 4B, and the second equality from the fact $X_t^\top \mathbf{a}_{t+1} =$

$X_t^\top X_t \left(X_t^\top X_t\right)^\dagger \mathbf{x}_{t+1} = \mathbf{x}_{t+1}$.

By the assumption made in the theorem statement, $\left|\mathbf{a}_{t+1}^\top \mathbf{e}_t\right|$ decreases to 0 quickly at the rate of $\mathcal{O}(t^{-\alpha})$, which means $t = \Omega(\epsilon^{-1/\alpha})$ will be large enough to guarantee $\left|\mathbf{a}_{t+1}^\top \mathbf{e}_t\right| \leq \epsilon/2$. This fact indicates that when t is large enough (but still is polynomial in $\frac{1}{\epsilon}$, the least-squares prediction $\bar{y}_{t+1}$ has a prediction bias of $\xi + \epsilon/2$. This argument essentially converts the problem into one where the errors $e_t^* = 0$, and the theorem follows from a similar analysis to Strehl and Littman [2008b].

8.5.6 Proof of Theorem 28

Before proving the sample complexity of exploration bound, we first provide two useful lemmas.

Lemma 49 *Let f_1 and f_2 be two real-valued functions on the same finite domain X; namely, $f_i : X \mapsto \mathbb{R}$, for $i = 1, 2$. If $\max_{x \in X} |f_1(x) - f_2(x)| \leq \Delta$ for some $\Delta > 0$, then $|\max_{x \in X} f_1(x) - \max_{x \in X} f_2(x)| \leq \Delta$.*

PROOF. Define $x_i = \arg\max_{x \in X} f_i(x)$ for $i = 1, 2$, and then $\max_{x \in X} f_1(x) - \max_{x \in X} f_2(x) = f_1(x_1) - f_2(x_2)$. On the one hand, we have

$$f_1(x_1) - f_2(x_2) \geq f_1(x_2) - f_2(x_2) \geq -\Delta;$$

on the other hand, we have

$$f_1(x_1) - f_2(x_2) \leq f_1(x_1) - f_2(x_1) \leq \Delta;$$

The lemma is proved by combining these two inequalities. □

Lemma 50 *Let π be a policy for an H-horizon MDP. Let s_1 be a fixed start state of an episode, and s_h be the state visited at stage h of this episode. Then,*

$$V_1^*(s_1) - V_1^\pi(s_1) = \mathbf{E}_\pi \left[\sum_{h=1}^{H} \left(Q_h^*(s_h, \pi^*(s_h, h)) - Q_h^*(s_h, \pi(s_h, h)) \right) \right],$$

where $\mathbf{E}_\pi$ stands for the expectation with respect to the probability distributions of trajectories $\rho = [s_1, s_2, \cdots, s_H, s_{H+1}]$ generated by policy π.

PROOF. We let r_h denote the reward received at stage h by following π. Note that both s_h and r_h are random variables whose distributions are completely determined by policy π as s_1 is fixed. Then,

$$\begin{aligned} V_1^*(s_1) &= Q_1^*(s_1, \pi^*(s_1, 1)) \\ &= Q_1^*(s_1, \pi(s_1, 1)) + \Big(Q_1^*(s_1, \pi^*(s_1, 1)) - Q_1^*(s_1, \pi(s_1, 1))\Big) \\ &= \mathbf{E}_\pi \left[r_1 + V_2^*(s_2)\right] + \Big(Q_1^*(s_1, \pi^*(s_1, 1)) - Q_1^*(s_1, \pi(s_1, 1))\Big). \end{aligned}$$

We apply the derivation above for $V_h^*(s_h)$ recursively up to stage H, and obtain

$$V_1^*(s_1) = \mathbf{E}_\pi[r_1 + r_2 + \cdots + r_H] + \mathbf{E}_\pi \left[\sum_{h=1}^{H} \Big(Q_h^*(s_h, \pi^*(s_h, h)) - Q_h^*(s_h, \pi(s_h, h))\Big)\right].$$

By definition, $V_1^\pi(s_1) = \mathbf{E}_\pi[r_1 + r_2 + \cdots + r_H]$, which completes the proof. □

We are now ready to prove Theorem 28 by mathematical induction. For $h = H$, the theorem is ensured by Assumption 5. For the induction step, assume the theorem holds for all stages $l > h$ where $h < H$ and we consider stage h. Due to operations of Algorithm 8.4.3, the transitions from s_h to s_{h+1} in all episodes can be categorized into two groups: (i) $L_{h+1} = \text{TRUE}$, and (ii) $L_{h+1} = \text{FALSE}$.

Transitions belonging to case (i) consist of a stream of data for $\mathbf{A}_0^{(h)}$ to run according to the KWIK online regression protocol, and Assumption 5 guarantees that there are at most $B_0(d, 1/\epsilon_h, 1/\delta_h)$ timesteps for which $\perp$ is outputted. On the other hand, by the induction hypothesis, case (ii) happens at most $\sum_{l=h+1}^{H} B_0(d, 1/\epsilon_l, 1/\delta_l)$ times. Therefore, the total number of $\perp$ outputted in stage h is at most

$$B_0(d, 1/\epsilon_h, 1/\delta_h) + \sum_{l=h+1}^{H} B_0(d, 1/\epsilon_l, 1/\delta_l),$$

which is what we desire to prove for part I.

Part II follows directly from part I.

For part III, the target function to learn at stage h is given by

$$\tilde{Q}_h(s, a) = R(s, a) + \sum_{s' \in S} T(s, a, s') \max_{a' \in A} \hat{Q}_{h+1}(s', a'),$$

where $\hat{Q}_{h+1}$ is the function learned by REKWIRE in stage $h+1$.[7] By the induction hypothesis, we have $\left|\hat{Q}_{h+1}(s',a') - Q^*_{h+1}(s',a')\right| \leq H\sum_{l=h+1}^{H}(\epsilon_l+\xi_l)$ for all (s',a') whenever $\perp$ is not outputted. Let $\hat{Q}_h$ be the function $\mathbf{A}_0^{(h)}$ learns, then for any (s,a) we have $\left|\hat{Q}_h(s,a) - \tilde{Q}_h(s,a)\right| \leq H(\epsilon_h+\xi_h)$ due to Assumptions 5 and 6. Combining all these facts, we have for any (s,a):

$$\begin{aligned}
\left|\hat{Q}_h(s,a) - Q^*_h(s,a)\right| &\leq \left|\hat{Q}_h(s,a) - \tilde{Q}_h(s,a)\right| + \left|\tilde{Q}_h(s,a) - Q^*_h(s,a)\right| \\
&\leq H(\epsilon_h+\xi_h) + \left|\sum_{s'\in S} T(s,a,s')\Big(\max_{a'\in A}\hat{Q}_{h+1}(s',a') - \max_{a'\in A} Q^*_{h+1}(s',a')\Big)\right| \\
&\leq H(\epsilon_h+\xi_h) + \max_{s'\in S}\left|\max_{a'\in A}\hat{Q}_{h+1}(s',a') - \max_{a'\in A} Q^*_{h+1}(s',a')\right| \\
&\leq H(\epsilon_h+\xi_h) + H\sum_{l=h+1}^{H}(\epsilon_l+\xi_l) \\
&= H\sum_{l=h}^{H}(\epsilon_l+\xi_l),
\end{aligned}$$

where the last inequality is due to Lemma 49.

8.5.7 Proof of Theorem 29

For episode i, let p_i be the probability of entering some state s for which $\perp$ is outputted, when start states are drawn from μ_0. Denote by π_i the policy used by REKWIRE in episode i. Let $\hat{Q}_h$ be the value-function estimate of the algorithm for stage h.

Consider any state trajectory $\rho = [s_1, s_2, \cdots, s_H, s_{H+1}]$ generated by policy π_i. Two situations can occur: (i) $\perp$ is outputted (maybe multiple times) in ρ, and (ii) $\perp$ is not outputted in ρ. The probabilities of cases (i) and (ii) are p and $1-p$, respectively. When case (ii) happens, with probability at least $1-\sum_{h=1}^{H}\delta_h = 1-\frac{\delta}{2}$, we have for each

[7] Strictly speaking, $\hat{Q}_{h+1}$ may change over time and thus $\tilde{Q}_h$ is not a stationary learning target. But, this fact does not affect our analysis as long as $\hat{Q}_{h+1}$ is always bounded between $Q^*_{h+1} - H\sum_{l=h+1}^{H}(\epsilon_l+\xi_l)$ and $Q^*_{h+1} + H\sum_{l=h+1}^{H}(\epsilon_l+\xi_l)$.

h,

$$\begin{aligned}
& Q_h^*(s_h, \pi^*(s_h,h)) - Q_h^*(s_h, \pi_i(s_h,h)) \\
&\leq \; Q_h^*(s, \pi^*(s_h,h)) - \hat{Q}_h(s_h, \pi_i(s_h,h)) + \mathcal{O}\left(H^2(\epsilon_h + \xi_h)\right) \\
&= \; Q_h^*(s, \pi^*(s_h,h)) - \hat{Q}_h(s_h, \pi_i(s_h,h)) + \mathcal{O}\left(\frac{\epsilon}{H}\right) \\
&\leq \; Q_h^*(s, \pi^*(s_h,h)) - \hat{Q}_h(s_h, \pi^*(s_h,h)) + \mathcal{O}\left(\frac{\epsilon}{H}\right) \\
&\leq \; \mathcal{O}\left(H^2(\epsilon_h + \xi_h)\right) + \mathcal{O}\left(\frac{\epsilon}{H}\right) \\
&= \; \mathcal{O}\left(\frac{\epsilon}{H}\right), \qquad (8.7)
\end{aligned}$$

where the first and last inequalities are due to Corollary 2(III), and the second due to the fact that π_i is greedy with respect to $\hat{Q}_h$ when no $\perp$ is outputted.

For any fixed start state s_1, Lemma 50 asserts that

$$V_1^*(s_1) - V_1^{\pi_i}(s_1) = \mathbf{E}_{\pi_i}\left[\sum_{h=1}^{H}\left(Q_h^*(s_h, \pi^*(s_h,h)) - Q_h^*(s_h, \pi_i(s_h,h))\right)\right].$$

Combined with Equation (8.7) and the fact that case (i) happens with probability p_i, the equality above implies $V_1^*(s_1) - V_1^{\pi_i}(s_1) = \mathcal{O}(\epsilon + Hp_i)$. When $p_i \leq p_0$ for some threshold $p_0 = \mathcal{O}(\frac{\epsilon}{H})$, we have $V_1^*(s_1) - V_1^{\pi_i}(s_1) = \mathcal{O}(\epsilon)$ and also $\mathbf{E}_{s_1 \sim \mu_0}\left[V^*(s_1) - V^{\pi_i}(s_1)\right] = \mathcal{O}(\epsilon)$, indicating that the policy π_i is indeed $\mathcal{O}(\epsilon)$-optimal.

We claim that with high probability $p_i > p_0$ will hold only a polynomial number of episodes. Specifically, Corollary 2 asserts that $\perp$ is outputted $\mathcal{O}(H^2 B_0(d, \frac{H^3}{\epsilon}, \frac{H}{\delta}))$ times. Using the inequality of Hoeffding [1963], with probability at least $1 - \frac{\delta}{2}$, the number of episodes i with $p_i > p_0$ is

$$\mathcal{O}\left(\frac{H^2 B_0(d, \frac{H^3}{\epsilon}, \frac{H}{\delta})}{p_0} \ln \frac{1}{\delta}\right).$$

Substituting $p_0 = \mathcal{O}(\frac{\epsilon}{H})$ and applying the union bound to the two cases ($p_i > p_0$ and $p_i \leq p_0$) gives the lemma.

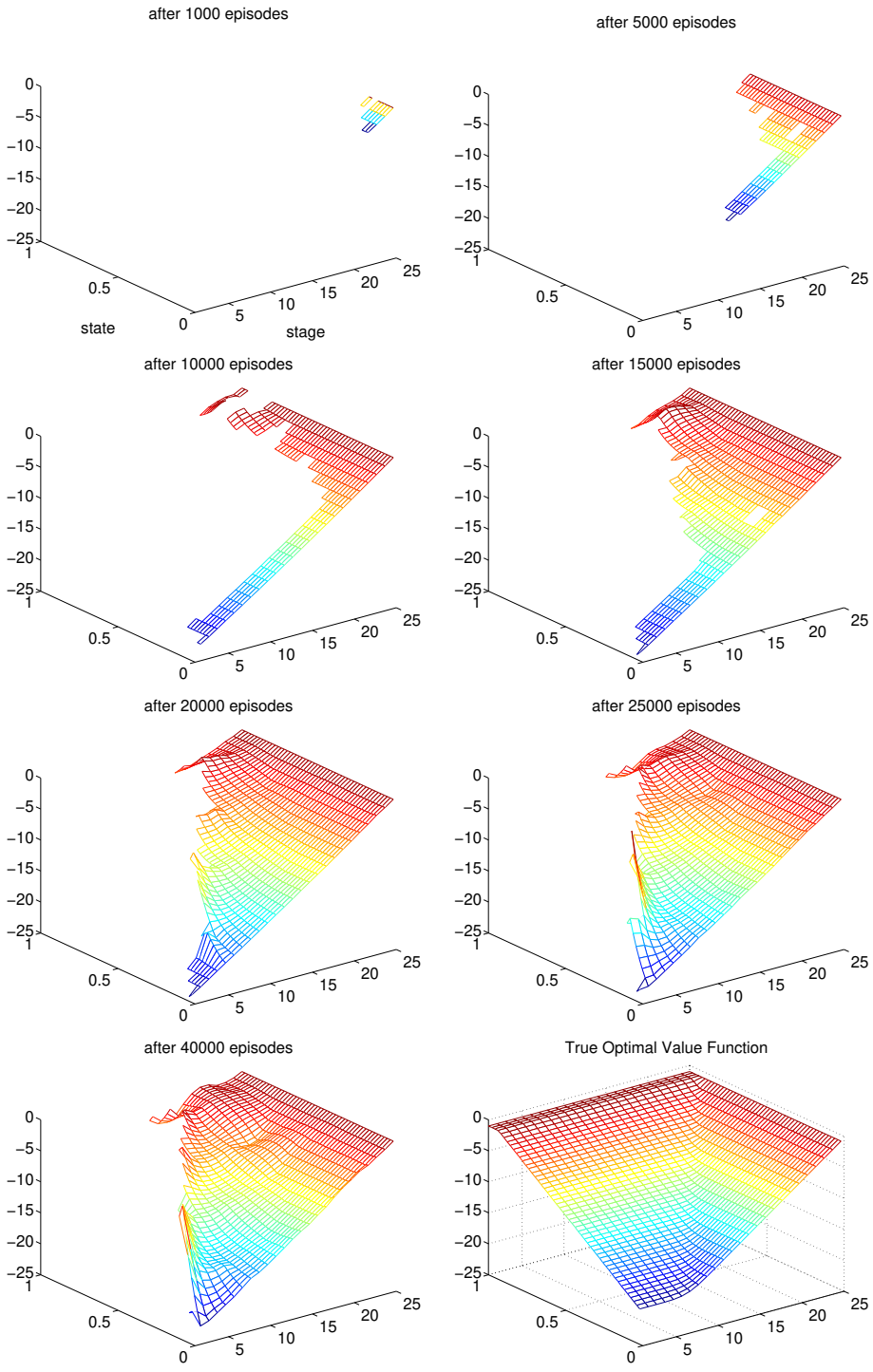


Figure 8.9: Value function evolution when RBF features are used. The bottom-right plot gives the true V^* function.

Part IV

Conclusions

Chapter 9

Conclusions

This chapter revisits the contributions of the dissertation, and then explains how they can be used to address the three challenges in reinforcement learning that are defined at the beginning of the dissertation. A number of open problems and possible extensions are discussed briefly.

9.1 Contributions Revisited

As mentioned in the introduction, the dissertation studies provably efficient exploration in sequential decision making problems where function approximation can be used. Here, we review the major contributions in the context of the three challenges defined in §1.2 and discuss how they help address these challenges.

First, the KWIK framework is developed to facilitate analyzing and developing algorithms for learning problems where active exploration can impact the training data the learner is exposed to. One of the key features in a KWIK learner is the option of explicitly saying "I don't know" (instead of making a normal prediction), which signals a need for exploration. This feature is essential for sequential decision making problems like reinforcement learning, in which a sub-optimal decision at one timestep may take a long time to recover from. This effect is best illustrated in the simple combination-lock example (§4.3.3), where a sub-optimal action can take $\Theta(|S|)$ steps to recover from, and the uninformative, random-walk-like ϵ-greedy rule ends up being exponentially inefficient in exploring this problem. Furthermore, KWIK is flexible enough to allow generalization between input–output pairs, thus providing opportunities to tackle large-scale problems. For instance, met-Rmax (§7.3.1) results in an exponential reduction in the sample complexity of exploration compared to Rmax by using a factored

representation to generalize over different states. As another example, CORL (§7.3.2) KWIK-learns a multivariate normal distribution and thus explores an infinitely large MDP with provably small sample complexity.

Second, the KWIK framework provides a unifying language for analyzing PAC-MDP algorithms. This tool is particularly helpful in the family of model-based algorithms, where the KWIK-Rmax algorithm (§7.1) unifies essentially all model-based PAC-MDP algorithms in the literature as well as the analyses of their sample complexity of exploration. Specifically, these include finite MDPs [Kearns and Singh 2002; Brafman and Tennenholtz 2002; Kakade 2003; Strehl et al. 2006a], metric MDPs [Kakade et al. 2003], factored-state MDPs [Kearns and Koller 1999; Strehl 2007a; Strehl et al. 2007; Diuk et al. 2009], MDPs with relocatable action models [Leffler et al. 2007], MDPs with linear dynamics [Strehl and Littman 2008b], MDPs with normal offsets [Brunskill et al. 2008], relational MDPs with object-oriented representations [Diuk et al. 2008], and MDPs with delayed observations [Walsh et al. 2009a]. We expect more KWIK-learnable classes of MDPs with practical interest can be found and treated in this algorithmic framework.

Third, KWIK-based PAC-MDP algorithms can also suggest algorithmic ideas for efficient exploration in existing algorithms, even if they are not PAC-MDP in general. The LSPI-Rmax algorithm combines ideas from Rmax with the powerful LSPI algorithm, and so tends to explore efficiently while using linear function approximation to avoid the curse of dimensionality. Similarly, one of the key features in delayed Q-learning is that it acts greedily with respect to an optimistic value function at all time. It may be interesting to investigate heuristics for making the value function optimistic in regular Q-learning and Sarsa, which has not been investigated in this dissertation.

Finally, delayed Q-learning and the matching lower bound of sample complexity proved in §8.2.4 sheds some light on the long-standing question of in what way estimating a model may help reinforcement learning. Our new lower bound is tight for the factors $|\mathcal{S}|$, $1/\epsilon$, and $1/\delta$, in the weaker *parallel sampling* model [Kearns and Singh 1999] (*c.f.*, §2.4). This finding suggests that a worse dependence on $1/\epsilon$ is possible only

in MDPs with slow *mixing* rates.[1] In both the parallel-sampling model and the MDP used to prove the lower bound in §8.2.4, the distribution of states being sampled/visited mixes extremely fast (in one and two timesteps, respectively). The slower the mixing rate, the more difficult is the *temporal credit assignment* problem [Sutton and Barto 1998]. In other words, a worse dependence on $1/\epsilon$ seems to require the construction of an MDP where *deep planning* is necessary.

Another important lesson learned from the study of delayed Q-learning is that it is unnecessary to learn an accurate model to explore efficiently in online reinforcement learning. Here, the sample complexity of exploration in this algorithm, $\tilde{\mathcal{O}}(|\mathcal{S}|)$, is smaller than the number of parameters needed to represent a transition probability function, $\mathcal{O}(|\mathcal{S}|^2)$. In contrast, all existing model-based PAC-MDP algorithms rely on accurately learning the MDP model, a process that can be expensive and difficult in practice.

9.2 Open Problems

We briefly discuss a few important open problems, some of which are motivated by the limitations of the work in this dissertation.

9.2.1 Issues in the KWIK Framework

Despite our focus on reinforcement learning here, the KWIK framework might find applications in other areas of machine learning. An in-depth study is interesting and may follow a similar line of study of the PAC and MB models.

An extremely important open problem is agnostic KWIK, which undoubtedly is difficult in general (*c.f.*, §5.5). This finding is not surprising, given the hardness results in agnostic PAC learning [Kearns et al. 1994], but the goal may be achievable in special situations.

---

[1] There are many ways to define a mixing rate. Roughly speaking, it measures how fast the distribution of states an agent reaches becomes independent of the initial state and the policy being followed.

Another open problem is to find a dimensionality measure for characterizing complexity of hypothesis classes that is useful in the KWIK context. A number of dimensionality measures have been developed in the literature [Angluin 2004] but do not appear to be directly usable in KWIK.

Last, but not least, it is useful to investigate techniques for converting a KWIK algorithm in deterministic problems to stochastic problems. Moving from the deterministic setting to the stochastic one, for example, the KWIK bound changes from $\mathcal{O}(k)$ to $\mathcal{O}(k \ln k/\epsilon^2)$ for union (§6.5), and from $\mathcal{O}(n)$ to $\tilde{\mathcal{O}}(n/\epsilon^4)$ for linear regression. Are there general-purpose conversion techniques?

9.2.2 Issues in Reinforcement Learning

Following the discussion of KWIK in the previous subsection, a natural question to ask in the reinforcement-learning context is: is KWIK necessary for provably efficient reinforcement learning? Either proving or disproving this statement require a careful formulation of the problem.

Second, it would be interesting to develop an KWIK-based, model-free algorithm that can be instantiated to different algorithms when different function approximations are used. Our result in §8.1 applies to finite MDPs and special cases of abstract-state MDPs only.

Third, we note that the main results in this dissertation are *worst-case* results: we consider the most-difficult-to-solve MDPs and devise conservative algorithms to handle such worse cases even if they may rarely happen in reality. In practice, however, this kind of exploration is inefficient and risky. Prior knowledge may step in and help. One form of prior knowledge is a non-trivial optimistic initialization of value functions, as is done by Asmuth et al. [2008] and Strehl et al. [2009]. Another form is a Bayesian prior distribution of possible world models, which has shown some benefits in balancing conservative (but provably efficient) exploration and prior knowledge about the model [Asmuth et al. 2009].

Finally, we note that the KWIK model we use here requires a hard separation between known and unknown labels. It is possible to extend many ideas to "soft"

version of knownness using techniques such as interval estimation [Strehl and Littman 2005; 2008a].

9.3 Concluding Remarks

Reinforcement learning possesses a unique combination of challenges that are fundamental to artificial intelligence and machine learning, including: exploration/exploitation tradeoff, sequential decision making, and generalization, among others. The present dissertation studies a novel computational learning framework, and uses it to analyze, unify, and create provably efficient algorithms that addresses these challenges. I hope this framework can provide a solid foundation for formal studies of reinforcement-learning algorithms, which eventually leads to creation of powerful, highly intelligent agents and practically useful solutions for real-life problems.

Part V

Appendices

Appendix A

Notation

This section summarizes the notation used frequently in the dissertation.

A.1 Fonts

In general, we use different fonts for different types of objects, as is summarized by Table A.1. However, we allow a few exceptions for compliance with convention. These exceptions are defined in the next sections.

Object Type	Font	Examples
action	slanted	*TurnRight*
algorithm	bold-face, upper case	$\mathbf{A}, \mathbf{A}_i$
algorithm name	sans-serif	KWIK-Rmax, delayed Q-learning
domain	small capitals	DIALER, BYCICLE
function	italic, lower case	f, g_i
matrix	italic, upper case	$A, B_t, \Sigma, \hat{\Sigma}$
operator	frankfurt, upper case	$\mathfrak{A}, \mathfrak{B}$
scalar	italic, lower case	a, i, j, t, x, y, z
set	calligraphic, upper case	$\mathcal{D}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{H}$
vector	bold-face, lower case	$\mathbf{x}, \mathbf{x}_t, \mathbf{w}, \boldsymbol{\mu}$
vector component	italic, lower case w/subscript	x_i, x_{tj}, w_k, μ_l

Table A.1: Fonts used in the dissertation for various objects.

A.2 Mathematical Notation

Table A.2 summarizes the notation for mathematics used in the dissertation. For completeness, we give precise definitions for some of the more important ones:

1. $\mathbb{I}(\cdot)$ is the indicator function: $\mathbb{I}(E) = 1$ if event E happens, and 0 otherwise.

Symbol	Meaning
$\emptyset$	empty set
$\mathbb{B}$	set of binary numbers $\{0, 1\}$
$\mathbb{N}$	set of natural numbers
$\mathbb{R}$	set of real numbers
$\mathbb{R}_+$	set of non-negative real numbers
$\mathbb{R}_-$	set of non-positive real numbers
$\mathcal{P}_{\mathcal{X}}$	set of probability distributions over some set $\mathcal{X}$
$\mathcal{Y}^{\mathcal{X}}$	set of functions mapping $\mathcal{X}$ to $\mathcal{Y}$
$[a_t]_{t\in\mathbb{N}}$	a sequence whose t-th element is a_t
$A = [a_{ij}]_{ij}$	matrix whose elements is a_{ij} in the (i,j)-entry
$\mathbf{0}_n$	the n-dimensional zero vector
I_n	the $n \times n$ identity matrix
O_{nm}	the $n \times m$ zero matrix
$\mathbb{I}(\cdot)$	set-indicator function
$\det A$	determinant of a square matrix A
$\mathrm{tr}\,(A)$	trace of a square matrix A (*i.e.*, the sum of its diagonal elements)
$\mathrm{d_{KL}}\,(\cdot,\cdot)$	Kullback-Leibler (KL) divergence between two distributions
$\mathrm{d_{var}}\,(\cdot,\cdot)$	total variation between two distributions
$\mathbf{E}[\cdot]$	expectation of a random variable
$\mathbf{Var}[\cdot]$	variance of a random variable
$\|\cdot\|_p$	vector or matrix ℓ_p-norm

Table A.2: Mathematical notation.

2. $\|\cdot\|_p$ (for $p > 0$) is the ℓ_p-norm that may be used for a vector $\mathbf{x} \in \mathbb{R}^n$, a matrix $A \in \mathbb{R}^{n\times m}$, or a real-valued function $f \in \mathbb{R}^{\mathcal{X}}$:

$$\begin{aligned} \|\mathbf{x}\|_p &\stackrel{\text{def}}{=} \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \\ \|A\|_p &\stackrel{\text{def}}{=} \max_{\mathbf{y}\in\mathbb{R}^m, \|\mathbf{y}\|_p=1} \|Ay\|_p \\ \|f\|_p &\stackrel{\text{def}}{=} \left(\int_{x\in\mathcal{X}} |f(x)|^p \,\mathrm{d}x\right)^{1/p}, \end{aligned}$$

assuming the integral above is well-defined. The ℓ_2-norm is sometimes called the Euclidean norm.

3. Asymptotic notation: Let f, g be two positive-valued functions defined on positive reals, then

(a) $f = \mathcal{O}(g)$ if there exist positive constants c and X such that one of the following happens (depending on which case is of interest): (i) $f(x) \leq cg(x)$ for all $x > X$, or (ii) $f(x) \leq cg(x)$ for $x \in (0, X)$;

(b) $f = o(g)$ if $g = \mathcal{O}(f)$;

(c) $f = \Theta(g)$ if both $f = \mathcal{O}(g)$ and $g = \mathcal{O}(f)$; and

(d) $f = \Omega(g)$ if there exist positive constants c and X such that one of the following happens (depending on which case is of interest): (i) $f(x) \geq cg(x)$ for all $x > X$, or (ii) $f(x) \geq cg(x)$ for $x \in (0, X)$;

A.3 Machine Learning and Reinforcement Learning Notation

Table A.3 summarizes notation used for machine learning and reinforcement learning in the dissertation.

Symbol	Meaning
$\mathcal{X}$	input set of a supervised-learning problem
$\mathcal{Y}$	output set of a supervised-learning problem
$\mathcal{Z}$	observation set of a supervised-learning problem
$\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$	hypothesis class of a supervised-learning problem
$\hat{f}$	estimate of an (unknown) quantity f
$\mathcal{A}$	set of actions in a Markov decision process
a, a_t	actions in a Markov decision process
$\mathfrak{A}$	approximation operator
$\mathfrak{B}, \mathfrak{B}^a$	Bellman operators
$\gamma \in [0, 1]$	discount factor in a Markov decision process
M, $\hat{M}$	Markov decision process
$\pi \in \mathcal{A}^{\mathcal{S}}$	a deterministic policy
$\pi \in (\mathcal{P}_{\mathcal{A}})^{\mathcal{S}}$	a stochastic policy
$R \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$	reward function in a Markov decision process
$\mathcal{S}$	set of states in a Markov decision process
$s, s_t \in \mathcal{S}$	states in a Markov decision process
$T \in (\mathcal{P}_{\mathcal{S}})^{\mathcal{S} \times \mathcal{A}}$	transition probabilities in a Markov decision process

Table A.3: Notation for machine learning and reinforcement learning.

Appendix B

Some Mathematical Facts

This chapter lists a number of mathematical facts that are used in the dissertation.

B.1 Statistics

Lemma 51 *(Union bound; Boole's inequality) If event E_i happens with probability $p_i \in [0,1]$, for $i = 1, 2, \ldots$, then the probability that at least one of these events happens is*

$$\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) \leq \sum_{i=1}^{\infty} p_i,$$

and the probability that none *of these events happens is*

$$\Pr\left(\bigcap_{i=1}^{\infty} \overline{A_i}\right) \geq 1 - \sum_{i=1}^{\infty} p_i,$$

where $\overline{A_i}$ denotes the complement of event A_i.

In Lemmas 52–55, a sequence of m random variables, $[x_i]_{1 \leq i \leq m}$, are involved. We use $\hat{\mu}$ to denote the empirical average:

$$\hat{\mu} \stackrel{\text{def}}{=} \frac{1}{m} \sum_{i=1}^{m} x_i,$$

Lemma 52 *[Hoeffding 1963] Let $x_1, x_2, \ldots, x_m$ be m independent random variables such that $L_i \leq x_i \leq U_i$ and $\mathbf{E}[x_i] = \mu_i$ for all i. The expectation of the empirical average, $\hat{\mu}$, is*

$$\mu \stackrel{\text{def}}{=} \mathbf{E}\left[\hat{\mu}\right] \;=\; \frac{1}{m} \sum_{i=1}^{m} \mu_i.$$

Then,

$$\begin{aligned} \Pr\left(\hat{\mu} - \mu \geq \epsilon\right) &\leq \exp\left(-\frac{2m^2\epsilon^2}{\sum_{i=1}^{m}(U_i - L_i)^2}\right), \\ \Pr\left(\hat{\mu} - \mu \leq -\epsilon\right) &\leq \exp\left(-\frac{2m^2\epsilon^2}{\sum_{i=1}^{m}(U_i - L_i)^2}\right). \end{aligned}$$

It is important to note that the sequence of random variables in Lemma 52 can in fact be martingales instead of independent random variables [Azuma 1967]. Such a fact is used in the dissertation.

A similar inequality concentration is given by the next lemma. It is sometimes called a multiplicative form of Chernoff's inequality.

Lemma 53 *[Chernoff 1952] Let $x_1, x_2, \ldots, x_m \in \mathbb{B}$ be m independent Bernoulli trials, each with a success probability μ: $\mathbf{E}[x_i] = \mu$ for all $i = 1, 2, \ldots, m$. Then for any $\alpha \in [0, 1]$, we have*

$$\Pr\left(\hat{\mu} > (1+\alpha)\mu\right) \leq \exp\left(-\frac{m\mu\alpha^2}{3}\right),$$
$$\Pr\left(\hat{\mu} < (1-\alpha)\mu\right) \leq \exp\left(-\frac{m\mu\alpha^2}{2}\right).$$

Lemma 54 *(Chebyshev's Inequality) Let $x_1, x_2, \ldots, x_m$ be m independent random variables such that $\mathbf{Var}[x_i] = \sigma^2$ and $\mathbf{E}[x_i] = \mu$ for all i. Then*

$$\Pr\left(|\hat{\mu} - \mu| \geq \epsilon\right) \leq \frac{\sigma^2}{m\epsilon^2}.$$

Lemma 55 *(Bernstein's Inequality) Let $x_1, x_2, \ldots, x_m$ be m independent random variables such that for all i, $\mathbf{E}[x_i] = \mu$, $\mathbf{Var}(x_i) = \sigma^2$, and*

$$\mathbf{E}\left[\left|x_i - \mu\right|^k\right] \leq \frac{\sigma^2 k! c^{k-2}}{2} \tag{B.1}$$

for some constant $c \in \mathbb{R}_+$ and all $k > 3$. Then

$$\Pr\left(|\hat{\mu} - \mu| \geq \epsilon\right) \leq \exp\left(-\frac{m\epsilon^2}{2(\sigma^2 + c\epsilon)}\right).$$

The next lemma is derived from Lemma 53 and improves a previous result [Strehl 2007b, Lemma 1]. The dissertation uses this improved result to get tighter analyses for many algorithms.

Lemma 56 *Let $x_1, x_2, x_3, \ldots \in \mathbb{B}$ be a sequence of m independent Bernoulli trials, each with a success probability at least μ: $\mathbf{E}[x_i] \geq \mu$, for some constant $\mu > 0$. Then for any $k \in \mathbb{N}$ and $\delta \in (0, 1)$, with probability at least $1 - \delta$, $x_1 + x_2 + \cdots + x_m \geq k$ if*

$$m \geq \frac{2}{\mu}\left(k + \ln\frac{1}{\delta}\right). \tag{B.2}$$

PROOF. Lemma 53 implies

$$\Pr\left(\sum_{i=1}^{m} x_i < k\right) \leq \exp\left(-\frac{m\mu\alpha^2}{2}\right),$$

where $k \stackrel{\text{def}}{=} (1-\alpha)m\mu$, and so $\alpha = 1 - \frac{k}{m\mu}$. To guarante the failure probability on the left-hand side is less than δ, it suffices to set m so that the right-hand side is no greater than δ, resulting in

$$m \geq \frac{2}{\mu\alpha^2} \ln\frac{1}{\delta} = \frac{2\ln\frac{1}{\delta}}{\mu\left(1-\frac{k}{m\mu}\right)^2}. \tag{B.3}$$

We want to find a large enough value for m to satisfy the inequality above. Using simple algebra, the inequality above can be rewritten as

$$m - \sqrt{\frac{2m}{\mu}\ln\frac{1}{\delta}} - \frac{k}{\mu} \geq 0.$$

Solving this quadratic equation for the unknown value $\sqrt{m}$ gives

$$\sqrt{m} \geq \sqrt{\frac{1}{2\mu}\ln\frac{1}{\delta}} + \sqrt{\frac{1}{2\mu}\ln\frac{1}{\delta} + \frac{k}{\mu}}.$$

Finally, using the elementary inequality, $(x+y)^2 \leq 2(x^2+y^2)$, we find the value of m given in Equation B.2 suffices to gaurantee Equation B.3. □

Lemma 57 *(Multidimensional Chebyshev's Inequality) Let $\mathbf{x} \sim \mathbb{R}^n$ be a random vector such that $\mathbf{E}[\mathbf{x}_i] = \mu \in \mathbb{R}^n$ and $\mathbf{E}[(\mathbf{x}-\mu)(\mathbf{x}-\mu)^\top] = \Sigma \in \mathbb{R}^{n\times n}$. Then*

$$\Pr\left(\sqrt{(\mathbf{x}-\mu)^\top\Sigma(\mathbf{x}-\mu)} \geq \epsilon\right) \leq \frac{n}{\epsilon^2}.$$

Lemma 58 *[Kullback 1967] Let p_1 and p_2 be two probability density functions defined over a space $\mathcal{X}$. Define $\mathcal{X}^+ = \{x \in \mathcal{X} \mid p_1(x) \geq p_2(x)\}$. If p_1 and p_2 are both measurable over $\mathcal{X}^+$, then*

$$\mathrm{d}_{\mathrm{KL}}(p_1, p_2) \geq \frac{1}{8}\|p_1 - p_2\|_1^2 = \frac{1}{2}\left(\mathrm{d}_{\mathrm{var}}(p_1, p_2)\right)^2.$$

B.2 Normal Distributions

We have used the following technical lemmas in the analysis of normal-learning (§6.6). While some of the facts are known (such as from the Wikipedia pages for normal

distributions[1] and multivariate normal distributions[2]), it is not easy to find a citation or a proof for them. Thus, we also provide complete proofs here. We start with a simple lemma involving the first four moments of a univariate normal distribution.

Lemma 59 *Let $x \sim \mathcal{N}(\mu, \sigma^2)$ be drawn from a univariate normal distribution. Then,*

$$\begin{aligned}
\mathbf{E}\left[x^2\right] &= \sigma^2 + \mu^2 \\
\mathbf{E}\left[x^3\right] &= \mu^3 + 3\mu\sigma^2 \\
\mathbf{E}\left[x^4\right] &= \mu^4 + 6\mu^2\sigma^2 + 3\sigma^4.
\end{aligned}$$

PROOF. The proof is a direct calculation through repeated applications of the integration-by-parts technique. The second moment is easy to compute from a well-known fact in probability theory:

$$\mathbf{E}\left[x^2\right] = \mathbf{Var}[x] + (\mathbf{E}[x])^2 = \sigma^2 + \mu^2.$$

For the third moment, we have:

$$\begin{aligned}
\mathbf{E}\left[x^3\right] &= \frac{1}{\sqrt{2\pi}\sigma}\int_{\mathbb{R}} x^3 \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\mathrm{d}x \\
&= \frac{1}{\sqrt{2\pi}\sigma}\int_{\mathbb{R}} x^2(x-\mu)\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\mathrm{d}x + \frac{\mu}{\sqrt{2\pi}\sigma}\int_{\mathbb{R}} x^2\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\mathrm{d}x \\
&= \left[\frac{-\sigma^2}{\sqrt{2\pi}\sigma}x^2\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\right]_{-\infty}^{\infty} + \frac{2\sigma^2}{\sqrt{2\pi}\sigma}\int_{\mathbb{R}} x\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\mathrm{d}x + \mu\mathbf{E}\left[x^2\right] \\
&= 0 + 2\sigma^2\mathbf{E}[x] + \mu\mathbf{E}\left[x^2\right] \\
&= \mu^3 + 3\mu\sigma^2,
\end{aligned}$$

Similarly,

$$\begin{aligned}
\mathbf{E}\left[x^4\right] &= \frac{1}{\sqrt{2\pi}\sigma}\int_{\mathbb{R}} x^4 \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\mathrm{d}x \\
&= \frac{1}{\sqrt{2\pi}\sigma}\int_{\mathbb{R}} x^3(x-\mu)\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\mathrm{d}x + \frac{\mu}{\sqrt{2\pi}\sigma}\int_{\mathbb{R}} x^3\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\mathrm{d}x \\
&= \left[\frac{-\sigma^2}{\sqrt{2\pi}\sigma}x^3\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\right]_{-\infty}^{\infty} + \frac{3\sigma^2}{\sqrt{2\pi}\sigma}\int_{\mathbb{R}} x^2\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\mathrm{d}x + \mu\mathbf{E}\left[x^3\right] \\
&= 0 + 3\sigma^2\mathbf{E}\left[x^2\right] + \mu\mathbf{E}\left[x^3\right] \\
&= \mu^4 + 6\mu^2\sigma^2 + 3\sigma^4.
\end{aligned}$$

[1] *http://en.wikipedia.org/wiki/Normal_distribution.*

[2] *http://en.wikipedia.org/wiki/Multivariate_normal_distribution.*

□

Lemma 60 *Let $x \sim \mathcal{N}(\mu, \sigma^2)$ be drawn from a univariate normal distribution. Then, for any $p \in \mathbb{N}$,*

$$\mathbf{E}\left[|x-\mu|^{2p}\right] = \frac{(2p)!\sigma^{2p}}{2^p p!}.$$

PROOF. Using the change-of-variable technique with $y = (x-\mu)/\sigma$, we have

$$\begin{aligned}\mathbf{E}\left[|x-\mu|^{2p}\right] &= \frac{1}{\sqrt{2\pi}\sigma}\int_{\mathbb{R}} |x-\mu|^{2p} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \mathrm{d}x \\ &= \frac{\sigma^{2p}}{\sqrt{2\pi}}\int_{\mathbb{R}} y^{2p} \exp\left(-\frac{y^2}{2}\right) \mathrm{d}y. \end{aligned} \tag{B.4}$$

Define

$$f(p) \stackrel{\text{def}}{=} \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}} y^{2p} \exp\left(-\frac{y^2}{2}\right) \mathrm{d}y.$$

Using the integration-by-parts technique, we may compute $f(p)$ by

$$\begin{aligned} f(p) &= \left[-\frac{y^{2p-1}}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right)\right]_{-\infty}^{\infty} + \frac{2p-1}{\sqrt{2\pi}}\int_{\mathbb{R}} y^{2(p-1)} \exp\left(-\frac{y^2}{2}\right) \mathrm{d}y \\ &= (2p-1)f(p-1). \end{aligned}$$

Applying the above equality recursively, we have

$$f(p) = (2p-1)f(p-1) = \cdots = (2p-1)(2p-3)\cdots 5\cdot 3\cdot f(1).$$

Observe that $f(1)$ is in fact the second moment of the normally distributed random variable, y, and thus Lemma 59 implies $f(1) = 1$. Consequently,

$$\begin{aligned} f(p) &= (2p-1)(2p-3)\cdots 5\cdot 3\cdot 1 \\ &= \frac{(2p)(2p-1)(2p-2)(2p-3)\cdots 3\cdot 2\cdot 1}{(2p)(2p-2)\cdots 4\cdot 2} \\ &= \frac{(2p)!}{2^p p!}. \end{aligned}$$

The lemma follows immediately by substituting the above formula of $f(p)$ in Equation B.4. □

Lemma 61 *Let $\mathcal{N}(\mathbf{0}, I_n)$ be an n-dimensional multivariate normal distribution, and $\Sigma = [\sigma_{ij}]_{ij} \in \mathbb{R}^{n\times n}$ a matrix. Then,*

$$\mathbf{E}_{\mathbf{x}\sim\mathcal{N}(\mathbf{0},I_n)}\left[\mathbf{x}^\top \Sigma \mathbf{x}\right] = \operatorname{tr}(\Sigma).$$

PROOF. Denote $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^\top$. Then

$$\begin{aligned}
\mathbf{E}_{\mathbf{x}\sim\mathcal{N}(\mathbf{0},I_n)}\left[\mathbf{x}^\top\Sigma\mathbf{x}\right] &= \frac{1}{(2\pi)^{n/2}}\int_{\mathbb{R}^n}\mathbf{x}^\top\Sigma\mathbf{x}\exp\left(-\frac{\mathbf{x}^\top\mathbf{x}}{2}\right)\mathrm{d}\mathbf{x} \\
&= \frac{1}{(2\pi)^{n/2}}\int_{\mathbb{R}^n}\sum_{i=1}^n\sum_{j=1}^n\sigma_{ij}\mathbf{x}_i\mathbf{x}_j\exp\left(-\frac{\mathbf{x}^\top\mathbf{x}}{2}\right)\mathrm{d}\mathbf{x} \\
&= \frac{1}{(2\pi)^{n/2}}\sum_{i=1}^n\sum_{j=1}^n\sigma_{ij}\int_{\mathbb{R}^n}\mathbf{x}_i\mathbf{x}_j\exp\left(-\frac{\mathbf{x}^\top\mathbf{x}}{2}\right)\mathrm{d}\mathbf{x} \\
&= \frac{1}{2\pi}\sum_{i=1}^n\sum_{j\neq i}\sigma_{ij}\int_{\mathbb{R}^2}\mathbf{x}_i\mathbf{x}_j\exp\left(-\frac{\mathbf{x}_i^2+\mathbf{x}_j^2}{2}\right)\mathrm{d}\mathbf{x}_i\mathrm{d}\mathbf{x}_j \\
&\quad + \frac{1}{\sqrt{2\pi}}\sum_{i=1}^n\sigma_{ii}\int_{\mathbb{R}}\mathbf{x}_i^2\exp\left(-\frac{\mathbf{x}_i^2}{2}\right)\mathrm{d}\mathbf{x}_i.
\end{aligned}$$

The first term above must be 0, as for all $i \neq j$:

$$\int_{\mathbb{R}^2}\mathbf{x}_i\mathbf{x}_j\exp\left(-\frac{\mathbf{x}_i^2+\mathbf{x}_j^2}{2}\right)\mathrm{d}\mathbf{x}_i\mathrm{d}\mathbf{x}_j = \int_{\mathbb{R}}\mathbf{x}_i\exp\left(-\frac{\mathbf{x}_i^2}{2}\right)\mathrm{d}\mathbf{x}_i\int_{\mathbb{R}}\mathbf{x}_j\exp\left(-\frac{\mathbf{x}_j^2}{2}\right)\mathrm{d}\mathbf{x}_j = 0.$$

We can now complete the proof:

$$\mathbf{E}_{\mathbf{x}\sim\mathcal{N}(\mathbf{0},I_n)}\left[\mathbf{x}^\top\Sigma\mathbf{x}\right] = \frac{1}{\sqrt{2\pi}}\sum_{i=1}^n\sigma_{ii}\int_{\mathbb{R}}\mathbf{x}_i^2\exp\left(-\frac{\mathbf{x}_i^2}{2}\right)\mathrm{d}\mathbf{x}_i = \sum_{i=1}^n\sigma_{ii} = \mathrm{tr}\left(\Sigma\right),$$

where we have used Lemma 59 on individual normally distributed variable $\mathbf{x}_i$. □

Lemma 62 *Given two n-dimensional multivariate normal distributions, $\mathcal{N}(\mu,\Sigma)$ and $\mathcal{N}(\hat{\mu},\hat{\Sigma})$, such that Σ and $\hat{\Sigma}$ are non-singular, then*

$$\mathrm{d}_{\mathrm{KL}}\left(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\hat{\mu},\hat{\Sigma})\right) = \frac{1}{2}\left((\mu-\hat{\mu})^\top\hat{\Sigma}^{-1}(\mu-\hat{\mu}) + \ln\frac{\det\hat{\Sigma}}{\det\Sigma} + \mathrm{tr}\left(\hat{\Sigma}^{-1}\Sigma\right) - n\right).$$

PROOF. The proof is a direct calculation of the KL-divergence between these two normal distributions.

$$\begin{aligned}
&\mathrm{d_{KL}}\left(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\hat{\mu},\hat{\Sigma})\right)\\
&= \frac{1}{(2\pi)^{n/2}\sqrt{\det\Sigma}}\int_{\mathbb{R}^n}\exp\left(-\frac{(\mathbf{x}-\mu)^\top\Sigma^{-1}(\mathbf{x}-\mu)}{2}\right)\\
&\quad\cdot\ln\left(\sqrt{\frac{\det\hat{\Sigma}}{\det\Sigma}}\exp\left(\frac{(\mathbf{x}-\hat{\mu})^\top\hat{\Sigma}^{-1}(\mathbf{x}-\hat{\mu})-(\mathbf{x}-\mu)^\top\Sigma^{-1}(\mathbf{x}-\mu)}{2}\right)\right)\mathrm{d}\mathbf{x}\\
&= \frac{1}{2}\ln\frac{\det\hat{\Sigma}}{\det\Sigma}\frac{1}{(2\pi)^{n/2}\sqrt{\det\Sigma}}\int_{\mathbb{R}^n}\exp\left(-\frac{(\mathbf{x}-\mu)^\top\Sigma^{-1}(\mathbf{x}-\mu)}{2}\right)\mathrm{d}\mathbf{x}\\
&\quad-\frac{1}{2(2\pi)^{n/2}\sqrt{\det\Sigma}}\int_{\mathbb{R}^n}(\mathbf{x}-\mu)^\top\Sigma^{-1}(\mathbf{x}-\mu)\exp\left(-\frac{(\mathbf{x}-\mu)^\top\Sigma^{-1}(\mathbf{x}-\mu)}{2}\right)\mathrm{d}\mathbf{x}\\
&\quad+\frac{1}{2(2\pi)^{n/2}\sqrt{\det\Sigma}}\int_{\mathbb{R}^n}(\mathbf{x}-\hat{\mu})^\top\hat{\Sigma}^{-1}(\mathbf{x}-\hat{\mu})\exp\left(-\frac{(\mathbf{x}-\mu)^\top\Sigma^{-1}(\mathbf{x}-\mu)}{2}\right)\mathrm{d}\mathbf{x}.
\end{aligned}$$

Denote the three terms above by t_1, t_2 and t_3. Then, $\mathrm{d_{KL}}\left(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\hat{\mu},\hat{\Sigma})\right) = t_1 - t_2 + t_3$. First, it is easy to see that

$$t_1 = \frac{1}{2}\ln\frac{\det\hat{\Sigma}}{\det\Sigma}.$$

Second, we may change the variable by $\mathbf{x} = \Sigma^{1/2}\mathbf{y} + \mu$ and rewrite t_2 by

$$t_2 = \frac{1}{2(2\pi)^{n/2}}\int_{\mathbb{R}^n}\mathbf{y}^\top\mathbf{y}\exp\left(-\frac{\mathbf{y}^\top\mathbf{y}}{2}\right)\mathrm{d}\mathbf{y} = \frac{1}{2}\mathrm{tr}\left(I_n\right) = \frac{n}{2},$$

where Lemma 61 is used. Third, we use the same change of variable and rewrite t_3:

$$t_3 = \frac{1}{2(2\pi)^{n/2}}\int_{\mathbb{R}^n}\mathbf{y}^\top\mathbf{y}\exp\left(-\frac{\mathbf{y}^\top\Sigma^{1/2}\hat{\Sigma}^{-1}\Sigma^{1/2}\mathbf{y}}{2}\right)\mathrm{d}\mathbf{y} = \frac{1}{2}\mathrm{tr}\left(\Sigma^{1/2}\hat{\Sigma}^{-1}\Sigma^{1/2}\right),$$

where we use Lemma 61 again. Using Lemma 63, we have

$$t_3 = \frac{1}{2}\mathrm{tr}\left(\Sigma^{1/2}(\hat{\Sigma}^{-1}\Sigma^{1/2})\right) = \frac{1}{2}\mathrm{tr}\left((\hat{\Sigma}^{-1}\Sigma^{1/2})\Sigma^{1/2}\right) = \frac{1}{2}\mathrm{tr}\left(\hat{\Sigma}^{-1}\Sigma\right).$$

Combining the values of t_1, t_2, and t_3, we complete the proof of the lemma. □

B.3 Linear Algebra

Here, we list a few lemmas from linear algebra that are useful in the dissertation.

Lemma 63 *For any matrix $A = [a_{ij}]_{ij} \in \mathbb{R}^{n\times m}$ and matrix $B = [b_{ij}]_{ij} \in \mathbb{R}^{m\times n}$, the following is valid:*

$$\operatorname{tr}(AB) = \operatorname{tr}(BA).$$

PROOF. Using the definition of matrix trace and after simple algebra, we may see

$$\operatorname{tr}(AB) = \sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}b_{ji} = \operatorname{tr}(BA).$$

□

Lemma 64 *[Horn and Johnson 1986, Theorem 1.2.12] For any matrix $M \in \mathbb{R}^{n\times n}$, let $\psi_1, \psi_2, \ldots, \psi_n$ be its eigenvalues, then:*

$$\begin{aligned} \operatorname{tr}(A) &= \psi_1 + \psi_2 + \cdots + \psi_n, \\ \det A &= \psi_1\psi_2\cdots\psi_n. \end{aligned}$$

Lemma 65 *[Golub and Van Loan 1996, Lemma 2.7.1, Theorem 2.7.2] Suppose $Ax = b$ and $(A + \Delta A)y = b + \Delta b$ with $\|\Delta A\| \le \epsilon \|A\|$ and $\|\Delta b\| \le \epsilon \|b\|$. If $\epsilon\kappa(A) < 1$, then $A + \Delta A$ is nonsingular, and*

$$\frac{\|y - x\|}{\|x\|} \le \frac{2\epsilon\kappa(A)}{1 - \epsilon\kappa(A)},$$

where $\|\cdot\|$ can be any ℓ_p matrix/vector norm, and $\kappa(A) = \|A\| \left\|A^{-1}\right\|$ is the corresponding condition number*.*

Lemma 66 *[von Neumann 1937] Let $A, B \in \mathbb{R}^{n\times n}$ be two symmetric matrices, whose singular values are $a_1 \ge a_2 \ge \cdots \ge a_n \ge 0$ and $b_1 \ge b_2 \ge \cdots \ge b_n \ge 0$, respectively. Then,*

$$|\operatorname{tr}(AB)| \le \sum_{i=1}^{n} a_i b_i.$$

B.4 Miscellaneous

Lemma 67 *(Cauchy-Schwarz Inequality) Let u and v be two real-valued random variables. Their marginal and joint probability distributions are denoted by P_u, P_v, and P_{uv}, respectively. Then*

$$\mathbf{E}_{u,v\sim P_{xy}} |uv| \le \sqrt{\mathbf{E}_{u\sim P_u} u^2}\sqrt{\mathbf{E}_{v\sim P_v} v^2}.$$

Bibliography

Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML-05)*, pages 1–8, 2005.

Pieter Abbeel, Daphne Koller, and Andrew Y. Ng. Learning factor graphs in polynomial time and sample complexity. *Journal of Machine Learning Research*, 7:1743–1788, 2006a.

Pieter Abbeel, Margan Quigley, and Andrew Y. Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-06)*, pages 1–8, 2006b.

Naoki Abe, Alan W. Biermann, and Philip M. Long. Reinforcement learning with immediate rewards and linear hypotheses. *Algorithmica*, 37(4):263–293, 2003.

James S. Albus. A theory of cerebellar functions. *Mathematical Biosciences*, 10(1/2):25–61, 1971.

David Andre, Nir Friedman, and Ronald Parr. Generalized prioritized sweeping. In *Advances in Neural Information Processing Systems 10*, pages 1001–1007, 1998.

Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

Dana Angluin. Queries revisited. *Theoretical Computer Science*, 313(2):175–194, 2004.

András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with Bellman-residual minimizing based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

John Asmuth, Michael L. Littman, and Robert Zinkov. Potential-based shaping in model-based reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 604–609, 2008.

John Asmuth, Lihong Li, Michael L. Littman, Ali Nouri, and David Wingate. A Bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09)*, 2009.

Christopher G. Atkeson and Geoffrey J. Gordon, editors. *Proceedings of the ICML-97 Workshop on Modelling in Reinforcement Learning*, 1997. URL: `http://www.cs.cmu.edu/~ggordon/ml97ws`.

Christopher G. Atkeson and Juan Carlos Santamaria. A comparison of direct and model-based reinforcement learning. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation (ICRA-97)*, pages 3557–3564, 1997.

Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11(1–5):75–113, 1997a.

Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1–5):11–73, 1997b.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Advances in Neural Information Processing Systems 19 (NIPS-06)*, pages 49–56, 2007.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002a.

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002b.

Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal*, 19(3):357–367, 1967.

Leemon C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 30–37, 1995.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:835–846, 1983. Reprinted in J. A. Anderson and E. Rosenfeld, Neurocomputing: Foundations of Research, MIT Press, Cambridge, MA, 1988.

Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1–2):81–138, 1995.

Jonathan Baxter and Peter Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Andrey Bernstein and Nahum Shimkin. Adaptive aggregation for reinforcement learning with efficient exploration: Deterministic domains. In *Proceedings of the Twenty-First Annual Conference on Learning Theory (COLT-08)*, 2008.

Donald A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, 1985. ISBN 0412248107.

Dimitri P. Bertsekas. Convergence of discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, 20(3):415–419, 1975.

Dimitri P. Bertsekas. Distributed dynamic programming. *IEEE Transactions on Automatic Control*, AC-27(3):610–616, 1982.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1/2. Athena Scientific, 2nd edition, 2001. ISBN 1-886-52908-6.

Dimitri P. Bertsekas and David A. Castañon. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34(6):589–598, 1989.

Dimitri P. Bertsekas and Steven E. Shreve. *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific, 1996. ISBN 1-886529-03-5. Originally published in 1978 by Academic Press.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989. ISBN 0-13-648700-9. Reprinted by Athena Scientific in 1997.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, September 1996. ISBN 1-886529-10-8.

Avrim Blum. Separating distribution-free and mistake-bound learning models over the Boolean domain. *SIAM Journal on Computing*, 23(5):990–1000, 1994.

Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107, 2000.

Justin A. Boyan. Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, November 2002.

Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems 6 (NIPS-93)*, pages 671–678, 1994.

Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7 (NIPS-94)*, pages 369–376, 1995.

Justin A. Boyan and Andrew W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.

Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

Ronen I. Brafman and Moshe Tennenholtz. R-max—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, October 2002.

John L. Bresina, Richard Dearden, Nicolas Meuleau, Sailesh Ramkrishnan, David E. Smith, and Richard Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the Eighteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 77–84, 2002.

Emma Brunskill, Bethany R. Leffler, Lihong Li, Michael L. Littman, and Nicholas Roy. CORL: A continuous-state offset-dynamics reinforcement learner. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI-08)*, pages 53–61, 2008.

Emma Brunskill, Bethany R. Leffler, Lihong Li, Michael L. Littman, and Nicholas Roy. Provably efficient learning with typed parametric models, 2009. Conditionally accepted to the Journal of Machine Learning Research.

Nicolò Cesa-Bianchi, Philip M. Long, and Manfred Warmuth. Worst-case quadratic loss bounds for prediction using linear functions and gradient descent. *IEEE Transactions on Neural Networks*, 7(3):604–619, 1996.

Nicolò Cesa-Bianchi, Gábor Lugosi, and Gilles Stoltz. Minimizing regret with label efficient prediction. *IEEE Transactions on Information Theory*, 51(6):2152–2162, 2005.

Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Worst-case analysis of selective sampling for linear classification. *Journal of Machine Learning Research*, 7:1205–1230, 2006.

Hyeong Soo Chang, Michael C. Fu, Jiaqiao Hu, and Steven I. Marcus. An adaptive sampling algorithm for solving Markov decision processes. *Operations Research*, 53(1):126–139, 2005.

David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 726–731, 1991.

Ward Cheney and Will Light. *A Course in Approximation Theory*. Brooks/Cole Publishing Company, 2000. ISBN 0-534-36224-9.

Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

Chee-Seng Chow and John N. Tsitsiklis. The complexity of dynamic programming. *Journal of Complexity*, 5(4):466–488, 1989.

Chee-Seng Chow and John N. Tsitsiklis. An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control*, 36(8):898–814, 1991.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8 (NIPS-95)*, pages 1017–1023, 1996.

Daniela P. de Farias and Benjamin Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

Daniela Pucci de Farias and Benjamin Van Roy. On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478, 2004.

Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

Thomas Dean, Robert Givan, and Sonia M. Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 124–131, 1997.

F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1): 98–108, 1963.

Persi Diaconis and David Freedman. On the consistency of Bayes estimates. *The Annals of Statistics*, 14(1):1–67, 1986. With discussions.

Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 240–247, 2008.

Carlos Diuk, Lihong Li, and Bethany R. Leffler. The adaptive k-meteorologists problem and its application to structure discovery and feature selection in reinforcement learning. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML-09)*, 2009.

Michael O'Gordon Duff. *Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts, Amherst, MA, 2002.

Yaakov Engel, Shie Mannor, and Ron Meir. Bayes meets Bellman: The caussian process approach to temporal difference learning. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 154–161, 2003.

Eyal Even-Dar and Yishay Mansour. Approximate equivalence of Markov decision processes. In *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory (COLT-03)*, volume 2777 of *Lecture Notes in Computer Science*, pages 581–594, 2003a.

Eyal Even-Dar and Yishay Mansour. Learning rates for Q-learning. *Journal of Machine Learning Research*, 5:1–25, 2003b.

Eyal Even-Dar, Shie Mannor, and Yishay Mansour. PAC bounds for multi-armed bandit and Markov decision processes. In *Proceedings of the Fifteenth Annual Conference on Computational Learning Theory (COLT-02)*, pages 255–270, 2002.

A. A. Fel'dbaum. Dual-control theory. I. *Automation and Remote Control*, 21:874–880, 1961.

Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 162–169, 2004.

Claude-Nicolas Fiechter. Efficient reinforcement learning. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory (COLT-94)*, pages 88–97, 1994.

Yoav Freund, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. Using and combining predictors that specialize. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing (STOC-97)*, pages 334–343, 1997a.

Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2–3):133–168, 1997b.

Yoav Freund, Yishay Mansour, and Robert E. Schapire. Generalization bounds for averaged classifiers. *The Annals of Statistics*, 32(4):1698–1722, 2004.

Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3): 209–226, 1977.

Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modifications of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA, 2006.

Alborz Geramifard, Michael H. Bowling, and Richard S. Sutton. Incremental least-squares temporal difference learning. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, pages 356–361, 2006.

Alborz Geramifard, Michael H. Bowling, Martin Zinkevich, and Richard S. Sutton. iLSTD: Eligibility traces and convergence analysis. In *Advances in Neural Information Processing Systems 19 (NIPS-06)*, pages 441–448, 2007.

Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2–3): 323–390, 1992.

Robert Givan, Sonia Leach, and Thomas Dean. Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122(1–2):71–109, 2000.

Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1–2):163–223, 2003.

Sally A. Goldman and Michael J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, 1995.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996. ISBN 0-801-85414-8.

Geoffrey J. Gordon. Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 261–268, 1995.

Geoffrey J. Gordon. Chattering in Sarsa(λ). Technical report, Carnegie Mellon University Learning Lab, Pittsburgh, PA, 1996.

Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 1st edition, July 2003. ISBN 0387952845.

David P. Helmbold, Nick Littlestone, and Philip M. Long. Apple tasting. *Information and Computation*, 161(2):85–139, 2000.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1986. ISBN 0-521-38632-2.

Roland A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960. ISBN 0-262-08009-5.

Nicholas K. Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 752–757, 2005.

Nicholas K. Jong and Peter Stone. Model-based exploration in continuous state spaces. In *Proceedings of the Seventh International Symposium on Abstraction, Reformulation and Approximation (SARA-07)*, volume 4612 of *Lecture Notes in Computer Science*, pages 258–272, 2007.

Leslie Pack Kaelbling. Associative reinforcement learning: Functions in k-DNF. *Machine Learning*, 15(3):279–298, 1994.

Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. URL `http://citeseer.nj.nec.com/kaelbling96reinforcement.html`.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

Sham Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, UK, 2003.

Sham Kakade, Michael J. Kearns, and John Langford. Exploration in metric state spaces. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 306–312, 2003.

Konstantinos V. Katsikopoulos and Sascha E. Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE Transactions on Automatic Control*, 48(4): 568–574, 2003.

Michael J. Kearns and Daphne Koller. Efficient reinforcement learning in factored MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 740–747, 1999.

Michael J. Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993.

Michael J. Kearns and Robert E. Schapire. Efficient distribution-free learning of probabilistic concepts. *Journal of Computer and System Sciences*, 48(3):464–497, 1994.

Michael J. Kearns and Satinder P. Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. In *Advances in Neural Information Processing Systems 11 (NIPS-98)*, volume 11, pages 996–1002, 1999.

Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2–3):209–232, 2002.

Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, August 1994. ISBN 0262111934.

Michael J. Kearns, Robert E. Schapire, and Linda Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2-3):115–141, 1994.

Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2–3):193–208, 2002.

Philipp W. Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-06)*, pages 449–456, 2006.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML-06)*, pages 282–293, 2006.

Sven Koenig and Reid G. Simmons. The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, 22(1–3): 227–250, 1996.

J. Zico Kolter and Andrew Y. Ng. Near Bayesian exploration in polynomial time. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML-09)*, 2009a.

J. Zico Kolter and Andrew Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML-09)*, 2009b.

Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems 12 (NIPS-99)*, pages 1008–1014, 2000.

Vijay R. Konda and John N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2003.

George Konidaris and Sarah Osentoski. Value function approximation in reinforcement learning using the Fourier basis. Technical Report UM-CS-2008-19, Department of Computer Science, University of Massachusetts, Amherst, 2008.

Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2–3):189–211, 1990.

Solomon Kullback. A lower bound for discrimination in terms of variation. *IEEE Transactions on Information Theory*, 13(1):126–127, January 1967.

P.R. Kumar and Pravin Varaiya. *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice Hall, 1986. ISBN 013846684X.

Leonid Kuvayev and Richard S. Sutton. Approximation in model-based learning. In *Proceedings of the ICML-97 Workshop on Modelling in Reinforcement Learning*, 1997.

Michail G. Lagoudakis and Michael L. Littman. Algorithm selection using reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-00)*, pages 511–518, 2000.

Michail G. Lagoudakis and Michael L. Littman. Learning to select branching rules in the DPLL procedure for satisfiability. In *Workshop on Theory and Applications of Satisfiability Testing (SAT-01)*. Elsevier, 2001. Electronic Notes in Discrete Mathematics (ENDM) Volume 9.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003a.

Michail G. Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 424–431, 2003b.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.

John Langford and Satinder P. Singh. Reinforcement learning theory tutorial, 2006. In the Twenty-Third International Conference on Machine Learning (ICML-06).

Bethany R. Leffler, Michael L. Littman, Alexander L. Strehl, and Thomas J. Walsh. Efficient exploration with latent structure. In *Robotics: Science and Systems*, pages 81–88, 2005.

Bethany R. Leffler, Michael L. Littman, and Timothy Edmunds. Efficient reinforcement learning with relocatable action models. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, pages 572–577, 2007.

Lihong Li. Focus of attention in reinforcement learning. Master's thesis, University of Alberta, Edmonton, AB, Canada, 2004. Tech report TR07-12, Department of Computing Science, University of Alberta. URL: http://www.cs.ualberta.ca/research/techreports/.

Lihong Li. A worst-case comparison between temporal difference and residual gradient with linear function approximation. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 560–567, 2008.

Lihong Li and Michael L. Littman. Lazy approximation for solving continuous finite-horizon MDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 1175–1180, 2005.

Lihong Li and Michael L. Littman. Efficient value-function approximation via online linear regression. In *Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics (ISAIM-08)*, 2008a.

Lihong Li and Michael L. Littman. Prioritized sweeping converges to the optimal value function. Technical Report DCS-TR-631, Department of Computer Science, Rutgers University, 2008b.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics (ISAIM-06)*, 2006.

Lihong Li, Vadim Bulitko, and Russell Greiner. Focus of attention in reinforcement learning. *Journal of Universal Computer Science*, 13(9):1246–1269, 2007.

Lihong Li, Michael L. Littman, and Thomas J. Walsh. Knows what it knows: A framework for self-aware learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 568–575, 2008.

Lihong Li, Michael L. Littman, and Christopher R. Mansley. Online exploration in least-squares policy iteration. In *Proceedings of the Eighteenth International Conference on Agents and Multiagent Systems (AAMAS-09)*, 2009a.

Lihong Li, Michael L. Littman, Alexander L. Strehl, and Thomas J. Walsh. Knows what it knows: A framework for self-aware learning, 2009b. Working draft.

Lihong Li, Jason D. Williams, and Suhrid Balakrishnan. Reinforcement learning for spoken dialog management using least-squares policy iteration and fast feature selection, 2009c. Submitted.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321, 1992.

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithms. *Machine Learning*, 2(4):285–318, 1988.

Nick Littlestone. From on-line to batch learning. In *Proceedings of the Second Annual Workshop on Computational Learning Theory (COLT-89)*, pages 269–284, 1989.

Michael L. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, Providence, RI, 1996.

Michael L. Littman. Probabilistic propositional planning: Representations and complexity. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 748–754, 1997.

Michael L. Littman, Thomas Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 394–402, 1995.

Manuel Loth, Manuel Davy, and Philippe Preux. Sparse temporal difference learning using LASSO. In *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 352–359, 2007.

Omid Madani. On policy iteration as a Newton's method and polynomial policy iteration algorithms. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 273–278, 2002.

Omid Madani, Daniel J. Lizotte, and Russell Greiner. The budgeted multi-armed bandit problem. In *Proceedings of the Seventeenth Annual Conference on Learning Theory (COLT-04)*, pages 643–645, 2004.

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

Stéphe G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12), 1993.

Shie Mannor and John N. Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5:623–648, 2004.

Yishay Mansour and Satinder Singh. On the complexity of policy iteration. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 401–408, 1999.

Andrew McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, NY, 1995.

H. Brendan McMahan and Geoffrey J. Gordon. Fast exact planning in Markov decision processes. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 151–160, 2005.

Francisco S. Melo and M. Isabel Ribeiro. Q-learning with linear function approximation. In *Proceedings of the Twentieth Annual Conference on Computational Learning Theory (COLT-07)*, pages 308–322, 2007.

Francisco S. Melo, Sean P. Meyn, and M. Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 664–671, 2008.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.

Oliver Mihatsch and Ralph Neuneier. Risk-sensitive reinforcement learning. *Machine Learning*, 49(2–3):267–290, 2002.

W. Thomas Miller, Filson H. Glanz, and L. Gordon Kraft. CMAC: An associative neural network alternative to backpropagation. *Proceedings of the IEEE*, 78(10):1561–1567, 1990.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, March 1997. ISBN 0-07-042807-7.

Davaadorjin Monhor. A Chebyshev inequality for multivariate normal distribution. *Probability in the Engineering and Informational Sciences*, 21:289–300, 2007.

Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(3): 199–233, 1995.

David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

Rémi Munos. Error bounds for approximate policy iteration. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 560–567, 2003.

Rémi Munos. Performance bounds in l_p-norm for approximate value iteration. *SIAM Journal on Control and Optimization*, 46(2):541–561, 2007.

Rémi Munos and Andrew W. Moore. Rates of convergence for variable resolution schemes in optimal control. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-00)*, pages 647–654, 2000.

Rémi Munos and Andrew W. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49(2–3):291–323, 2002.

Rémi Munos and Csaba Szepesvári. Finite-time bounds for sampling-based fitted value iteration. *Journal of Machine Learning Research*, 9:815–857, 2008.

Andrew Y. Ng and Michael I. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 406–415, 2000.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*, pages 278–287, 1999.

Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16 (NIPS-03)*, 2004.

Ali Nouri and Michael L. Littman. Multi-resolution exploration in continuous spaces. In *Advances in Neural Information Processing Systems 21 (NIPS-08)*, pages 1209–1216, 2009.

Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine Learning*, 49: 161–178, 2002.

Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994. ISBN 0201530821.

Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael L. Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML-07)*, pages 737–744, 2007.

Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 752–759, 2008.

Jing Peng and Ronald J. Williams. Efficient learning and planning within the Dyna framework. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 281–290, 1993.

Theodore J. Perkins and Doina Precup. A convergent form of approximate policy iteration. In *Advances in Neural Information Processing Systems 15 (NIPS-02)*, pages 1595–1602, 2003.

Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML-07)*, pages 745–750, 2007.

Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Natural actor-critic. In *Proceedings of the Sixteenth European Conference on Machine Learning (ECML-05)*, pages 280–291, 2005.

Marek Petrik. An analysis of Laplacian methods for value function approximation in MDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2574–2579, 2007.

Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-06)*, pages 697–704, 2006.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, New York, 1994. ISBN 0-471-61977-9.

Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth Interational Conference on Machine Learning (ICML-98)*, pages 463–471, 1998.

Carl Edward Rasmussen and Malte Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems 16 (NIPS-03)*, 2004.

Balaraman Ravindran and Andrew G. Barto. SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1011–1016, 2003.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.

J. M. Robson. The complexity of Go. In *IFIP Congress*, pages 413–417, 1983.

David F. Rogers, Robert D. Plante, Richard T. Wong, and James R. Evans. Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, 39(4): 553–582, 1991.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, December 2002. ISBN 0-137-90395-2.

John Rust. Using randomization to break the curse of dimensionality. *Econometrica*, 65(3): 487–516, 1997.

Paul Ruvolo, Ian R. Fasel, and Javier R. Movellan. Optimization on a budget: A reinforcement learning approach. In *Advances in Neural Information Processing Systems 21 (NIPS-08)*, pages 1385–1392, 2009.

Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3(3):210–229, 1959.

Jeff G. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In *Advances in Neural Information Processing Systems 9 (NIPS-96)*, pages 1047–1053, 1997.

Ralf Schoknecht and Artur Merke. TD(0) converges provably faster than the residual gradient algorithm. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 680–687, 2003.

Dale Schuurmans and Lloyd Greenwald. Efficient exploration for optimizing immediate reward. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 385–392, 1999.

Paul J. Schweitzer and Abraham Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110(2):568–582, 1985.

Burr Settles. Active learning literature survey. Technical Report 1648, Department of Computer Sciences, University of Wisconsin-Madison, January 2009.

H. Sebastian Seung, Manfred Opper, and Naftali Tishby. Query by committee. In *Proceedings of the Fifth Annual Conference on Computational Learning Theory (COLT-92)*, pages 287–294, 1992.

Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9:371–421, 2008.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, June 2004. ISBN 0521813972.

David Silver, Richard S. Sutton, and Martin Müller. Reinforcement learning of local shape in the game of Go. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1053–1058, 2007.

Özgür Şimşek and Andrew G. Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-06)*, pages 833–840, 2006.

Satinder P. Singh and Dimitri P. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems 9 (NIPS-96)*, pages 974–980, 1997.

Satinder P. Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

Satinder P. Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3): 287–308, 2000.

Eduardo D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, volume 6 of *Texts in Applied Mathematics*. Springer, 2nd edition, July 1998. ISBN 0-387-98489-5.

Alexander L. Strehl. Model-based reinforcement learning in factored-state MDPs. In *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 103–110, 2007a.

Alexander L. Strehl. *Probably Approximately Correct (PAC) Exploration in Reinforcement Learning*. PhD thesis, Rutgers University, New Brunswick, NJ, 2007b.

Alexander L. Strehl and Michael L. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the Twenty-Second Conference on Machine Learning (ICML-05)*, pages 857–864, 2005.

Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008a.

Alexander L. Strehl and Michael L. Littman. Online linear regression and its application to model-based reinforcement learning. In *Advances in Neural Information Processing Systems 20 (NIPS-07)*, pages 1417–1424, 2008b.

Alexander L. Strehl, Lihong Li, and Michael L. Littman. Incremental model-based learners with formal learning-time guarantees. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI-06)*, pages 485–493, 2006a.

Alexander L. Strehl, Lihong Li, and Michael L. Littman. PAC reinforcement learning bounds for RTDP and Rand-RTDP. In *Proceedings of the AAAI-06 Workshop on Learning for Search*, 2006b. Also an AAAI technical report WS-06-11, pages 50–56.

Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-06)*, pages 881–888, 2006c.

Alexander L. Strehl, Chris Mesterharm, Michael L. Littman, and Haym Hirsh. Experience-efficient learning in associative bandit problems. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-06)*, pages 889–896, 2006d.

Alexander L. Strehl, Carlos Diuk, and Michael L. Littman. Efficient structure learning in factored-state MDPs. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 645–650, 2007.

Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite MDPs: PAC analysis, 2009. Conditionally accepted to the Journal of Machine Learning Research.

Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, MA, 1984.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

Richard S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Workshop on Machine Learning (ICML-90)*, pages 216–224, 1990.

Richard S. Sutton. Planning by incremental dynamic programming. In *Proceedings of the Eighth International Workshop on Machine Learning (ICML-91)*, pages 353–357, 1991.

Richard S. Sutton. Introduction: The challenge of reinforcement learning. *Machine Learning*, 8(3–4):225–227, 1992.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8 (NIPS-95)*, pages 1038–1044, 1996.

Richard S. Sutton. The reward hypothesis, 2004. http://rlai.cs.ualberta.ca/RLAI /rewardhypothesis.html.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, March 1998. ISBN 0-262-19398-1.

Richard S. Sutton, David McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12 (NIPS-99)*, pages 1057–1063, 2000.

Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael H. Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI-08)*, pages 528–536, 2008.

Csaba Szepesvári. The asymptotic convergence-rate of Q-learning. In *Advances in Neural Information Processing Systems 10 (NIPS-97)*, pages 1064–1070, 1998.

Csaba Szepesvári and Michael L. Littman. A unified analysis of value-function-based reinforcement-learning algorithms. *Neural Computation*, 11(8):2017–2059, 1999.

Csaba Szepesvári and William D. Smart. Interpolation-based Q-learning. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-04)*, pages 100–107, 2004.

Gavin Taylor and Ronald Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML-09)*, 2009.

Jonathan Taylor, Prakash Panangaden, and Doina Precup. Bounding performance loss in approximate mdp homomorphisms. In *Advances in Neural Information Processing Systems 21 (NIPS-08)*, pages 1649–1656, 2009.

Russ Tedrake, Teresa Weirui Zhang, Ming-fai Fong, and H. Sebastian Seung. Actuating a simple 3D passive dynamic walker. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-04)*, pages 4656–4661, 2004.

Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995.

Gerald Tesauro. Online resource allocation using decompositional reinforcement learning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 886–891, 2005.

Gerald Tesauro and Gregory R. Galperin. On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing Systems 9 (NIPS-96)*, pages 1068–1074, 1997.

Sebastian Thrun. The role of exploration in learning control. In David A. White and Donald A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, pages 527–559. Van Nostrand Reinhold, 1992.

Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1–2):99–141, 2001.

Robert Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society (Series B)*, 58(1):267–288, 1996.

John Tromp and Gunnar Farnebäck. Combinatorics of Go. In *Computers and Games*, LNCS 4630, pages 84–99. Springer, 2007.

John N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202, 1994.

John N. Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1–3):59–94, 1996.

John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690, 1997.

John N. Tsitsiklis and Benjamin Van Roy. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks*, 12(4):694–703, 2001.

William Uther. *Tree Based Hierarchical Reinforcement Learning*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2002.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

Benjamin Van Roy. Performance loss bounds for approximate value iteration with state aggregation. *Mathematics of Operations Research*, 31(2):234–244, 2006.

Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer, 2nd edition, March 2006. ISBN 0387308652. Reprint of 1982 Edition, including "Empirical Inference Science" (Afterword of 2006).

John von Neumann. Some matrix-inequalities and metrization of matrix-space. *Tomck University Review*, 1:286–300, 1937.

Thomas J. Walsh, Ali Nouri, Lihong Li, and Michael L. Littman. Planning and learning in environments with delayed feedback. In *Proceedings of the Eighteenth European Conference on Machine Learning (ECML-07)*, volume 4701 of *Lecture Notes in Computer Science*, pages 442–453, 2007.

Thomas J. Walsh, Ali Nouri, Lihong Li, and Michael L. Littman. Planning and learning in environments with delayed feedback. *Journal of Autonomous Agents and Multi-Agent Systems*, 18(1):83–105, 2009a.

Thomas J. Walsh, István Szita, Carlos Diuk, and Michael L. Littman. Exploring compact reinforcement-learning representations with linear regression. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09)*, 2009b.

Christopher J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, University of Cambridge, UK, 1989.

Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

Tsachy Weissman, Erik Ordentlich, Gadiel Seroussi, Sergio Verdu, and Marcelo J. Weinberger. Inequalities for the l_1 deviation of the empirical distribution. Technical Report HPL-2003-97, HP Lab, Palo Alto, 2003.

Steven D. Whitehead. Complexity and cooperation in Q-learning. In *Proceedings of the Eighth International Workshop on Machine Learning (ICML-91)*, pages 363–367, 1991.

Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, 2006.

Jason D. Williams. *Partially Observable Markov Decision Processes for Spoken Dialogue Management*. PhD thesis, Cambridge University, Cambridge, UK, 2006.

Jason D. Williams. Integrating expert knowledge into POMDP optimization for spoken dialog systems. In *Proceedings of the AAAI-08 Workshop on Advancements in POMDP Solvers*, 2008. Also available in AAAI Technical Report WS-08-01.

Ronald J. Williams and Leemon C. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, College of Computer Science, Northeastern University, 1993.

David Wingate and Kevin D. Seppi. Prioritization methods for accelerating MDP solvers. *Journal of Machine Learning Research*, 6:851–881, 2005.

Xin Xu, Han gen He, and Dewen Hu. Efficient reinforcement learning using recursive least-squares methods. *Journal of Artificial Intelligence Research*, 16:259–292, 2002.

Xin Xu, Tao Xie, Dewen Hu, and Xicheng Lu. Kernel least-squares temporal difference learning. *International Journal of Information and Technology*, 11(9):54–63, 2005.

Xin Xu, Dewen Hu, and Xicheng Lu. Kernel-based least-squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007.

Hengshuai Yao and Zhi-Qiang Liu. Preconditioned temporal difference learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 1208–1215, 2008.

Wei Zhang and Thomas G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1114–1120, 1995.

Vita

Lihong Li

EDUCATION

Oct. 2009 Ph.D. in Computer Science, Rutgers University, New Brunswick, NJ, USA

Jul. 2004 M.Sc. in Computing Science, University of Alberta, Edmonton, AB, Canada

Jul. 2002 B.E. in Computer Science and Technology, Tsinghua University, Beijing, China

EXPERIENCE

01/2005—05/2009	Graduate Assistant, Department of Computer Science, Rutgers University, New Brunswick, NJ, USA
06/2008—08/2008	Research Intern, AT&T Shannon Labs, Florham Park, NJ, USA
05/2007—08/2007	Research Intern, Yahoo! Research, New York, NY, USA
05/2006—08/2006	Engineer Intern, Google Inc., New York, NY, USA
09/2002—07/2004	Teaching/Research Assistant, Department of Computing Science, University of Alberta, Edmonton, AB, Canada

PUBLICATION

1. Provably efficient learning with typed parametric models. Emma Brunskill, Bethany R. Leffler, **Lihong Li**, Michael L. Littman, and Nicholas Roy. Conditionally accepted to the *Journal of Machine Learning Research*.
2. Reinforcement learning in finite MDPs: PAC analysis. Alexander L. Strehl, **Lihong Li**, and Michael L. Littman. Conditionally accepted to the *Journal of Machine Learning Research*.
3. Maintaining equilibria during exploration in sponsored search auctions. John Langford, **Lihong Li**, Yevgeniy Vorobeychik, and Jennifer Wortman. *Algorithmica* (Special Issue on WINE-07), in print.
4. The adaptive k-meteorologists problem and its application to structure discovery and feature selection in reinforcement learning. Carlos Diuk, **Lihong Li** and Bethany R. Leffler. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML-09)*. Montreal, Canada, June, 2009.
5. A Bayesian sampling approach to exploration in reinforcement learning. John Asmuth, **Lihong Li**, Michael L. Littman, Ali Nouri, and David Wingate. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09)*. Montreal, Canada, June, 2009.
6. Sparse online learning via truncated gradient. John Langford, **Lihong Li**, and Tong Zhang. *Journal of Machine Learning Research*, 10:719–743, 2009.

7. Online exploration in least-squares policy iteration. **Lihong Li**, Michael L. Littman, and Christopher R. Mansley. In the *Eighteenth International Conference on Agents and Multiagent Systems (AAMAS-09)*. Budapest, Hungary, May, 2009.

8. Prioritized sweeping converges to the optimal value function. **Lihong Li** and Michael L. Littman. Technical report DCS-TR-631, Department of Computer Science, Rutgers University, 2008.

9. Sparse online learning via truncated gradient. John Langford, **Lihong Li**, and Tong Zhang. In *Advances in Neural Information Processing Systems 21 (NIPS-08)*, pages 905–912, 2009.

10. Planning and learning in environments with delayed feedback. Thomas J. Walsh, Ali Nouri, **Lihong Li**, and Michael L. Littman. *Journal of Autonomous Agents and Multi-Agent Systems*, 18(1):83–105, 2009.

11. A worst-case comparison between temporal difference and residual gradient with linear function approximation. **Lihong Li**. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 560–567. Helsinki, Finland, July, 2008.

12. Knows what it knows: A framework for self-aware learning. **Lihong Li**, Michael L. Littman, and Thomas J. Walsh. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 568–575. Helsinki, Finland, July, 2008. Co-winner of the "Best Student Paper" Award.

13. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. Ronald Parr, **Lihong Li**, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 752–759. Helsinki, Finland, July, 2008.

14. CORL: A continuous-state offset-dynamics reinforcement learner. Emma Brunskill, Bethany R. Leffler, **Lihong Li**, Michael L. Littman, and Nicholas Roy. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI-08)*, pages 53–61. Helsinki, Finland, July, 2008.

15. Efficient value-function approximation via online linear regression. **Lihong Li** and Michael L. Littman. In *Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics (AMAI-08)*. Fort Lauderdale, FL, January, 2008.

16. Maintaining equilibria during exploration in sponsored search auctions. Jennifer Wortman, Yevgeniy Vorobeychik, **Lihong Li**, and John Langford. In *Proceedings of the Third International Workshop on Internet and Network Economics (WINE-07)*, pages 119–130. San Diego, CA, December, 2007. Also appears in *Lecture Notes in Computer Science*, vol. 4858.

17. Focus of attention in reinforcement learning. **Lihong Li**, Vadim Bulitko, and Russell Greiner. In the *Journal of Universal Computer Science*, 13(9):1246–1269, November, 2007.

18. Planning and learning in environments with delayed feedback. Thomas J. Walsh, Ali Nouri, **Lihong Li**, and Michael L. Littman. In *Proceedings of the Eighteenth European Conference on Machine Learning (ECML-07)*, pages 442–453. Warsaw, Poland, September, 2007. Also appears in *Lecture Notes in Computer Science*, vol. 4701.

19. Analyzing feature generation for value-function approximation. Ronald Parr, Christopher Painter-Wakefield, **Lihong Li**, and Michael L. Littman. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML-07)*, pages 737–744. Corvallis, OR, June, 2007.

20. PAC model-free reinforcement learning. Alexander L. Strehl, **Lihong Li**, Eric Wiewiora, John Langford, and Michael L. Littman. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-06)*, pages 881–888. Pittsburgh, PA, June, 2006.

21. Incremental model-based learners with formal learning-time guarantees. Alexander L. Strehl, **Lihong Li**, and Michael L. Littman. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI-06)*, pages 485–493. Cambridge, MA, July, 2006.

22. Towards a unified theory of state abstraction for MDPs. **Lihong Li**, Thomas J. Walsh, and Michael L. Littman. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics (AMAI-06)*. Fort Lauderdale, FL, January, 2006.

23. Lazy approximation for solving continuous finite-horizon MDPs. **Lihong Li** and Michael L. Littman. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 1175–1180. Pittsburgh, PA, 2005.

24. Focus of attention in reinforcement learning. **Lihong Li**. *Master's thesis*, Department of Computing Science, University of Alberta, Edmonton, AB, Canada, July, 2004.

25. Batch reinforcement learning with state importance. **Lihong Li**, Vadim Bulitko, and Russell Greiner. In *Proceedings of the Fifteenth European Conference on Machine Learning (ECML-04)*, pages 566–568. Pisa, Italy, 2004. Also appears in *Lecture Notes in Computer Science*, vol. 3201.

26. Lookahead pathologies for single agent search. Vadim Bulitko, **Lihong Li**, Russell Greiner, and Ilya Levner. In *Proceedings of the Eighteenth International Joint Conferences on Artificial Intelligence (IJCAI-03)*, pages 1531–1533. Acapulco, Mexico, August, 2003.

27. Improving an adaptive image interpretation system by leveraging. **Lihong Li**, Vadim Bulitko, Russell Greiner, and Ilya Levner. In *Proceedings of the Eighth Australian and New Zealand Conference on Intelligent Information Systems*. Sydney, Australia, December 2003.

28. Learning robust object recognition strategies. Ilya Levner, Vadim Bulitko, **Lihong Li**, Greg Lee, and Russell Greiner. In *Proceedings of the Eighth Australian and New Zealand Conference on Intelligent Information Systems*. Sydney, Australia, December 2003.

29. Automated feature extraction for object recognition. Ilya Levner, Vadim Bulitko, **Lihong Li**, Greg Lee, and Russell Greiner. In *Proceedings of Image and Vision Computing'03 New Zealand*. Palmerston North, New Zealand, November 2003.

30. Towards automated creation of image interpretation systems. Ilya Levner, Vadim Bulitko, **Lihong Li**, Greg Lee, and Russell Greiner. In *Proceedings of the Sixteenth Australian Joint Conference on Artificial Intelligence*, pages 653–665. Perth, Australia, December 2003. Also appears in *Lecture Notes in Computer Science*, vol. 2903.

31. Typical XML document transformation methods and an application system. **Lihong Li**, Min Shao, Zhenkun Zheng, Chuan He, and Zhi-Hui Du. In *Computer Science*, 30(2):40–44, 2003. (In Chinese)

32. Design and implementation of an agent communication module based on KQML. **Lihong Li**. *Bachelor's thesis*, Department of Computer Science and Technology, Tsinghua University, Beijing, China, July, 2002.