Algorithms for Sequential Decision Making

Michael Lederman Littman
Ph.D. Dissertation

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-96-09
March 1996

Algorithms for Sequential Decision Making

by
Michael Lederman Littman
B.S., Yale University, May 1988
M.S., Yale University, May 1988

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University

May 1996

Abstract of “Algorithms for Sequential Decision Making”
by Michael Lederman Littman, Ph.D., Brown University, May 1996.

Sequential decision making is a fundamental task faced by any intelligent agent in an extended interaction with its environment; it is the act of answering the question “What should I do now?” In this thesis, I show how to answer this question when “now” is one of a finite set of states, “do” is one of a finite set of actions, “should” is maximize a long-run measure of reward, and “I” is an automated planning or learning system (agent). In particular, I collect basic results concerning methods for finding optimal (or near-optimal) behavior in several different kinds of model environments: Markov decision processes, in which the agent always knows its state; partially observable Markov decision processes (POMDPs), in which the agent must piece together its state on the basis of observations it makes; and Markov games, in which the agent is in direct competition with an opponent. The thesis is written from a computer-science perspective, meaning that many mathematical details are not discussed, and descriptions of algorithms and the complexity of problems are emphasized. New results include an improved algorithm for solving POMDPs exactly over finite horizons, a method for learning minimax-optimal policies for Markov games, a pseudopolynomial bound for policy iteration, and a complete complexity theory for finding zero-reward POMDP policies.

Vita

Michael Littman was born August 30th, 1966, in Philadelphia, Pennsylvania. He began working with computers when his parents bought him a TRS-80 for his 13th birthday and has not stopped since. After graduating from Plymouth-Whitemarsh High School in Plymouth-Meeting Pennsylvania in 1984, Michael attended Yale University. He graduated summa cum laude with B.S. and M.S. degrees in Computer Science in 1988, and was granted the computer science department's highest undergraduate honor. His Master's thesis, advised by M. Chen and performed in collaboration with C. Metcalf, discussed research on massively parallel computers.

Michael worked in T. K. Landauer's Cognitive Science Research Group at Bellcore for 4 years following graduation. He worked under the direct mentorship of D. Ackley, and collaborated extensively with several other researchers in the group. He co-authored many papers during this period, including papers on computer simulations of artificial life, social elements of human language understanding, an optimization algorithm for telephone switchport assignments, a system for automatically generating 3-dimensional views of data, and the computer analysis of bilingual text databases. The work on multilanguage analysis culminated in the joint issuing of U.S. patent 5,301,109 on April 5, 1994 with T. K. Landauer.

Michael was admitted to Bellcore's Support for Doctoral Education program, and began graduate school in 1992. He spent a year in Carnegie Mellon University's School for Computer Science and completed his graduate requirements before transferring to Brown University. By the time he began his work at Brown under L. P. Kaelbling, his research interests were fairly well developed. He co-authored a series of papers that formed the basis of his doctoral dissertation, including work on learning in Markov games, the complexity of algorithms for solving Markov decision processes, and a new algorithm for solving partially observable Markov decision processes.

Acknowledgments

It is possible that there are scientists that do not need the support and encouragement of other people to accomplish their goals; I am not such a scientist. I would like to acknowledge some of the people who have helped me do what I do, and have encouraged me to do it to the best of my ability. I extend my deepest gratitude to:

- My downward nuclear family. My spouse, Lisa, supports me and inspires me and makes my life more enjoyable by sharing hers with me. My son, Max, slept through more of my thesis than I hope anyone will do again. If he didn't have such a sweet personality and a strong constitution, I am sure that this thesis would have come out quite differently, if at all. I look forward to the day when he appreciates the irony of his name with respect to my research interests!

- My upward nuclear family. My parents, Howard and Phyllis, made it possible for me to say "Now I am a computer scientist" on my 13th birthday when they bought me my TRS-80—to this day, it is the only computer that I have ever owned. They taught me about striving and succeeding and working hard. During the preparation of this document, my mother completed her bachelor's degree and much of her master's degree, and my father taught himself how to surf the internet. Everything that I am began with them. My sister, Jill, and brother, Marc, put up with more than their share of my off-the-wall ideas when we were growing up. I am sure that my strong desire to be a teacher stems from the enjoyment I felt being their older brother.

- My in-law family. My mother-in-law, Gloria, father-in-law, David, and sister-in-law, Jenn, have always treated me with respect and love. The time that they spent caring for Max made it possible for me to work on some of the most difficult portions of this document. Their support was invaluable throughout this whole process; I only wish they could have earned "frequent driver miles" for schlepping back and forth to Rhode Island.

- My role models while growing up. Erwin Margolies, Phil Braun, Carol Dormuth, and others epitomized to me what it meant to be bright and well educated. I would not have thought of graduate school as a worthwhile goal if not for their influence.

- My college associates. In college, my classmates stimulated and inspired me, and continue to do so through their impressive accomplishments: Pratik Multani, Christopher Metcalf, Peter Schiffer, Aephraim Steinberg, Reyna Marder, Richard Katz, Steve Barkin, Riley Hart. I find myself hearing the wise words of my college professors whenever I try to teach myself something new: Richard Gerrig, Sandeep Bhatt, Alan Perlis.

- My friends and colleagues during my time at Bellcore. My fellow applied researchers always made me feel that I had something to contribute: Debby Swayne, Dan Ketchum, Tom Landauer, Mike Lesk, Mike Bianchi, Susan Dumais, Deborah Schmitt, Elaine Molchan. Dave Ackley's creativity, energy, openness, and presence astound everyone who knows him, and astound me even more because I feel I know him better than most. He was a powerful constructive force in my early research and I like to think of myself as his first graduating Ph.D. student. George Furnas could teach me simply by letting my watch him think; he is a good friend and an impressive role model. I believe that Rich Sutton, more than anyone, makes the field of reinforcement learning a nice place to work.

- My friends and colleagues throughout graduate school. I continue to turn to my CMU friends for their advice, encouragement, and skills: Justin Boyan, Shumeet Baluja, Geoff Gordon, Robert Driskill, Darrell Kindred, Lonnie Chrisman, Sven Koenig, Avrim Blum, Steven Rudich, Merrick Furst, Danny Sleator. My friends and colleagues at Brown helped my a great deal in carrying out this research: Jak Kirman, Mitch Cherniack, Eugene Charniak, Glenn Carroll, Ann Nicholson, Mike Brandstein (and his CTY friends), Kathy Kirman, Susan Platt, Dawn Nicholaus, José Castaños, Hagit Shatkay-Reshef, Sonia Leach, Jim Kurien, Shieu-Hong Lin.

- My support network. The women and babies of playgroup helped to broaden my perspective at a time when it was much easier to become overly focused: Lauren and Molly, Anne and Jonah, Tracey and Olivia, Sarah and Caitlin, Amy and Rachel, Kristen and Madeline, Miriam and Jared, Donna and Ross, Andrea and Benjamin, Betsy and Samantha, and Emily and James. Pearl Pena has given loving care to my family, especially Max; and Sarah Fallowes-Kaplan and Andy Kaplan contributed to my mental health and, to a lesser extent, my research. Justin Boyan is an excellent friend as well as a model-rival.

- My extended collection of research associates. Many people made direct contributions to my research through our discussions: Leemon Baird (extraneous solutions to the Bellman equation), Avrim Blum (the complexity of enumerating solutions to NP-hard problems), Justin Boyan (games and reinforcement learning), Anne Condon (solving alternating Markov games quickly), Stuart Geman (real analysis, types of convergence), Spike Hughes (generating random numbers over simplicies), Tommi Jaakkola (stochastic approximation), Daphne Koller (games and information), Harold Kushner (convergence issues), Andrew McCallum (tree-based methods for POMDPs), Ron Parr (solving information-state MDPs via reinforcement learning), Loren Platzman (POMDPs and finite-memory methods), Martin Puterman (quadratic convergence of policy iteration), John Rust (practical algorithms for MDPs), Ross Schachter (POMDPs, indirectly), Csaba Szepesvári (generalized reinforcement learning), John Tsitsiklis (complexity and applied math). Other colleagues were kind enough to respond to questions about their work via email: Craig Boutilier, Peter Dayan, Geoff Gordon, Mance Harmon, Matthias Heger, William Lovejoy, Jean-Luc Marion, Lisa Meeden, Mark Ring, Satinder Singh, David E. Smith, Benjiman Van Roy, Chip White.

- Those who contributed directly to this document. My thesis committee did their level best to keep me from putting my foot in my mouth: Tom Dean, Philip Klein, John Tsitsiklis. Tony Cassandra was my closest collaborator for this work; his competence and diligence made him an excellent collaborator, and will serve him well throughout his research career. Tony also deserves credit for creating the initial versions of several of the figures I used: Figures 2.1, 7.1, and especially 7.5. Leslie Kaelbling, my mentor and advisor, has taught me dozens of things about research and writing that I didn't even know that I didn't know. If I'm lucky, her influence will be felt in my work for a very long time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis document was submitted to the Graduate School at Brown University on February 27th, 1996. This technical report (version 2.0) contains only minor modifications from the original.

A frog jumps around a barrier to get to a delicious mealworm. A commuter tries an unexplored route to work and ends up having to stop and ask for directions. A major airline lowers prices for its overseas flights to try to increase demand. A pizza-delivery company begins a month-long advertising blitz. These are examples of sequential decision making; the purpose of this thesis is to explore automatic methods for choosing the best action in situations such as these.

The essence of sequential decision making is that decisions that are made now can have both immediate and long-term effects; the best choice to make now depends critically on future situations and how they will be faced. In this thesis, I explore algorithms, or automated procedures, for making the right decision in different sequential decision-making problems. Each chapter of the thesis presents a specific formal model and various algorithms for finding optimal behavior in that model. In the remainder of this chapter, I discuss sequential decision-making problems at a high level and explore what it means to solve them.

Some of the solution algorithms presented are extremely complicated. In the remainder of this section, I give some example sequential decision-making scenarios for which choosing the optimal or nearly optimal decision can be difficult; this should provide motivation for the importance of automatic sequential decision making, as well as help reveal why these problems require complicated algorithms. The examples are meant to be interesting problems worthy of additional study; none has yet been implemented as a large-scale system. For some example problems being solved in science and industry, along with estimates of how many millions of dollars their automated solutions are saving, see Puterman's recent textbook [126].

Budget Setting You are in charge of setting the budget for a large organization. Unfortunately, poor planning and bad management has left your organization with a huge amount of debt. It is your job to decide how to get the organization back in the black. You know that over the next few years some amount of spending must be cut, but cutting too much too soon will result in severe hardship for some members of your organization, while cutting too slowly will result in no budget balancing at all, because interest on the massive debt continues to compound.

Making things even more difficult are the effects on the budget of unpredictable factors, such as the precise amount of your organization's income over the next few years, its overall productivity, world-wide inflation, and the introduction of innovations that might help the organization save or even make money. On your side, you have detailed and accurate economic models for predicting the result of various budget-cutting options. The models can be used to reveal the probability that innovation will be curtailed by a 5% decrease in the budget, for example. Of course, simply knowing the probability of various events does not allow you to predict the future with certainty, but it gives you the ability to gauge the probability, over all possible futures, that the budget will eventually be balanced.

How can you use your knowledge of economic models and the current state of your organization to construct a budget-cutting plan for the next decade or so that brings down the debt with high probability as quickly as possible without destroying the organization in the process?

Baseball Pitching Now you are a professional pitcher facing a batter you know quite well. He hits .342, which makes him a high average hitter, but worse, he is well known for being a great home-run hitter. He is a lefty and you a righty, so you will not be able to get a curve ball to break away on him. There is one runner on second, and your team is ahead by 1 in the fifth inning.

A fast ball gives you the best chance of a quick strikeout, but it is likely that the batter is ready for it, and therefore that he has a good shot at hitting the ball out of the park. An inside slider might be a good idea, except that if it is off by even a little, he will belt it. What pitch do you throw to maximize your team's chances of winning the game, since you know the opposing batter will try to prevent you from winning?

Network Monitoring As an electrical engineer, it is your responsibility to design a piece of equipment that plugs into the telephone network to monitor and correct faults in a specific section of the network. The network consists of a set of switching computers and cables that connect the switching computers to one another. At any moment in time, the switching computers can be up or down and lightly or heavily loaded, and the cables can be transmitting or not transmitting information.

The monitoring equipment has access to various signals and alarms emitted by the switching computers, and it can send its own query signals to the switching computers to elicit feedback on their status. However, it is not possible to know with certainty the state of the network at any moment; for example, if a switching computer fails to respond to a query, is this because it is overloaded, because it has gone down, or because one of its incoming cables has stopped transmitting data?

In spite of the unreliable and sometimes inaccurate information, the monitoring equipment must keep the network running as smoothly as possible. To do this, it can send query and reset signals to the switching computers and can call for a repair person to physically examine any of the switching computers or cables. Of course, sending out a person can be expensive, especially late at night, but this expense is small when compared to the revenue lost due to the failure of a switching computer. Because of the difficulty in assessing when a switching computer has gone down, this can be an extremely difficult decision. How do you design the monitoring equipment to make this trade-off properly?

1.1 Sequential Decision Making

In this section, I begin to lay the groundwork for a formal model of sequential decision making. I describe the decision maker or agent, the environment with which it interacts, the behavior it exhibits, and the problems it might face.

1.1.1 The Agent

Through overuse, the word "agent" has come to mean very different things to different groups of people. In the context of this work, an agent is simply the system responsible for interacting with the world and making decisions. In the examples in the previous section, the agents are the budget director, the pitcher, and the network monitoring equipment. Agents can also be robots or software programs or medical equipment. The agents "live" in an environment: a particular economic model, a baseball game, a telephone network, the floor of an office building. The *state* of the environment is a description of everything that might change from moment to moment. In the baseball example, the state would include the position of the runners, the score, the inning, the identity of the current batter, and the number of balls and strikes. It probably would not include the color of the field, the number of bases, or the shape of home plate.

Figure 1.1 depicts a generic embedded agent [72] interacting with its environment. The agent is represented by a robotic figure and the environment as a blob. Although the agent is a decision-making entity, it is not enough for it simply to make decisions; it needs to turn these decisions into a selection of an *action* a to be taken to influence the state of the environment. The transition function T controls how actions alter the state s of the environment. In all but the most trivial environments, the agent's action choice should be a function of its perception z of the state. The component labeled O in the figure represents the agent's perception function, which transforms the environment state into a perception. For many environments, O is the identity function; that is, the agent has access to the true state of the environment. In others, such as in the network-monitoring example, the state of the environment is only partially observable to the agent. The function that maps perception to action choice is labeled in the figure with a B (for "behavior"). The component R is the agent's reward function and r the agent's reward; these are discussed in more detail in Section 1.1.3.

1.1.2 The Environment

Roughly, the environment is anything external to the agent. For the purposes of this thesis, I assume that environments are neither capricious nor malicious, but are instead oblivious of the agent and its goals. More concretely, I assume that the environment changes from state to state in response to the actions of the agent according to a fixed set of rules. The transitions might be stochastic; that is, it is not necessary that the

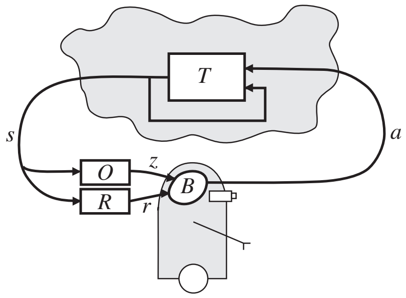

Figure 1.1: An embedded agent interacting with its environment.

same transition occur every time the agent takes a particular action in a particular state. However, the *probabilities* that govern these stochastic transitions must remain constant over time.

This definition of environment is somewhat restrictive and deserves a bit of elaboration. In most real-world problems, it is possible to identify the quickly changing aspects of the environment that constitute the state and the fixed aspects that are the environment. However, there are often aspects of the problem that change slowly over time. These can be modeled as part of the state, often with a large increase in complexity of the state description, or can be approximated as being static and made part of the fixed environment. A more accurate perspective would consider the environment to be non-stationary. Such environments are of interest because they make it possible to consider a broader range of sequential decision-making problems; however, they are more complex mathematically and few formal models have been proposed [42]. For this reason, previous attempts at handling non-stationary environments have focused almost exclusively on empirical studies [93, 156]. In this thesis, I am most concerned with the theoretical analysis of algorithms, and therefore restrict the discussion to environments that do not change over time.

Although I focus exclusively on time-invariant environments, I do not always assume that the agent has access to a complete environmental description. When this is the case, it is often possible for an agent to discover the time-invariant properties and exploit them; these problems are modeled in the reinforcement-learning framework described below.

Elements external to the agent that obey their own (not necessarily fixed) rules can sometimes be modeled as independent agents. For example, in the idealized baseball example, the players on the field follow the coaches' orders to the best of their abilities;

they are part of the environment. The opposing batter, however, is making decisions in a way that is purposely unpredictable and combative; for this reason, the environment can best be described as containing multiple agents.

1.1.3 Reward

To describe a sequential decision-making problem, it is not enough to specify the agent and the environment alone. The agent's actions need to serve some purpose: in the problems I consider, their purpose is to maximize *reward*. In a sense, reward is external to both the agent and the environment. It constitutes a specification of the problem the agent is to solve in the course of its interaction with the environment. In many sequential decision-making situations, there is a designer who can say for sure which actions and states are good and which are bad, and thus can explicitly specify the reward.

In the network-analyzer situation, for example, the rewards to the agent could be tied to the money spent and earned by the owners of the network: this information would be built into the agent by the designer. In the baseball example, the runs themselves would not be considered rewards, as they are not good or bad in and of themselves; the only true reward comes from winning the game. In the budget example, the reward criterion would need to be very complex and would probably be a source of considerable disagreement among the organization's management.

An important aspect of rewards is that they are the basis of the objective criterion used to judge agents' performance; however, for rewards to have any influence on behavior, agents must have subjective access to them. In the abstract models of environments considered in this thesis, rewards can be predicted by agents using their perception of the environment; this is a defining property of these models.

An interesting issue is whether there can be a notion of rewards in the "undesigned" environments faced by biological agents. In particular, the only true reward signal in a biological system is death, which is perceptible by the agent too late to be of use. Simulation experiments [3] have shown that, over the span of many generations, artificial agents can evolve their own proximal reward functions that are useful in predicting the relative goodness and badness of situations; in principle, even biological agents can compute their own reward functions. There is also evidence that specific structures in the brain of some animals are reward centers [110] that behave much like the reward functions used throughout this thesis. Of course, such analogies must not be taken too literally.

1.1.4 Policies

A policy is an agent's prescription for behavior. In general, an agent's policy is very complicated, including changes in behavior conditioned on events in the distant past. However, when the structure of the environment is known in advance, an agent can sometimes behave successfully with a much simpler policy.

A *plan* is the name for a particularly simple kind of policy in which the agent carries out a fixed sequence of actions "with its eyes closed;" that is, it takes the same sequence of actions regardless of what it perceives. Plans are important in completely predictable environments when the initial state of the agent is known in advance, as is the case, for example, with manufacturing robots.

A *conditional plan* admits a small amount of variation in the sequence of actions selected. For example, a part-painting robot might be built to work successfully even if it receives a part upside down. Its plan might have the form: gather part, check paint level in sprayer, request more paint if paint level too low, check orientation of part, use painting procedure A if part rightside up, use painting procedure B if part upside down.

An extreme form of conditional plan is a *stationary policy*, sometimes called a "universal plan" [138]. This type of policy has no specified sequence at all; instead, the agent examines the entire state at each step and then chooses the action most appropriate in the current state. For an agent to follow such a plan, it must have access to some function that returns an action choice for every possible state. A *partial policy* [43] can be used to overcome the difficulty of constructing and manipulating complicated universal plans; however, few theoretical tools are available to assess the success of a partial policy.

Stationary policies have several important properties that make them extremely important. First, in highly unpredictable environments, nearly any state can follow any other state, so any partial list of contingencies would be inadequate. Second, for the complex success criteria I discuss in this thesis, it is hard to imagine finding optimal behavior without reasoning about action choices in all possible states. Because of their central importance to algorithms for sequential decision making, we often use the word "policy" as an abbreviation for "stationary policy."

Policies can also be stochastic if the agent must flip a weighted coin to decide which action to take. This can be especially useful in competitive situations when it is important for the agent to be somewhat unpredictable; see Chapter 5.

1.1.5 Problem Scenarios

My purpose in this thesis is to examine methods for producing policies that maximize a measure of the long-run reward to an agent following it in a specific environment. These policies can be produced under two different problem scenarios that differ in the information available for constructing the policy: *planning* and *reinforcement learning*.

Planning In planning, a complete model of the environment is known in advance. This makes it possible to separate the decision-making problem into two components: the planner and the agent. The planner is responsible for taking a description of the environment and generating a policy, typically a stationary policy. This policy is then "downloaded" into the agent for execution in the environment. The task faced by the planner has a well-defined input and output, which makes it relatively easy to analyze from a traditional computer-science perspective. I undertake this type of analysis throughout the thesis.

Planning has been one of the primary subject areas in artificial intelligence since the development of the STRIPS system [53]. Early work on planning focused on the generation of plans for reaching some goal state in a deterministic environment. A more recent trend has been to consider decision-theoretic planning, in which more complex environmental models and optimality criteria are the norm. My interest in planning stems from this later work; I compare my work to relevant work in decision-theoretic planning in Section 8.1.

Reinforcement Learning The reinforcement-learning scenario is closely related to planning, although the two frameworks differ with respect to the information available about the environment. For a planner to function, it must have a complete description of the environment's states, actions, rewards, and transitions. Reinforcement learning can be used when a model of the environment is unknown or difficult to work with directly. The only access a reinforcement-learning agent has to information about its environment is via perception and action, making reinforcement learning a fairly

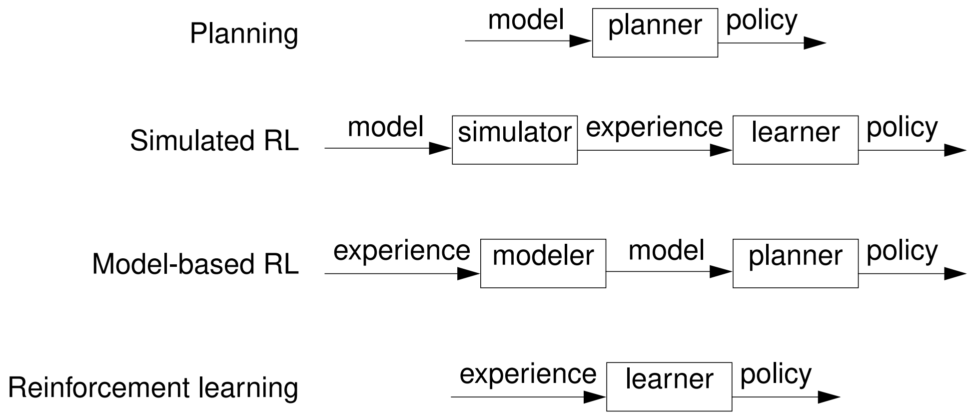

Figure 1.2: A collection of sequential decision-making scenarios.

difficult problem. Reinforcement learning removes the distinction between planner and agent: a reinforcement-learning agent is responsible for gathering information about the environment, organizing it to determine a policy, and behaving according to the policy.

An ideal reinforcement-learning agent chooses each action to maximize its long-term reward, perfectly selecting between actions to gain information about the environment and actions to gain reward. This goal is extremely difficult to achieve, and most reinforcement-learning algorithms focus on gathering information so that, over time, an optimal stationary policy can be generated. From this perspective, reinforcement learning and planning are closely related: a reinforcement learner carries out its planning in the context of direct interactions with the environment.

Hybrid Scenarios Although reinforcement learning is defined by the absence of an *a priori* model of the environment and planning by the presence of such a model, the techniques used to solve reinforcement-learning problems are useful for planning and vice versa. Figure 1.2 illustrates some of the various ways techniques can be combined to generate policies.

In *model-based reinforcement learning*, an agent uses its experience with the environment to construct an approximate model, which can be used as the input to a planning algorithm. This approach makes excellent use of available experience, which is often very expensive to gather, at the cost of invoking a full-fledged planning system whenever the agent needs to update its policy. An intermediate approach is to plan incrementally as new experience is encountered [155, 111, 119].

In *simulated reinforcement learning*, a reinforcement-learning agent is introduced into an environment with a known structure, but is forced to behave as if the structure is not known. Although this approach seems wasteful—how could throwing away information make decision making easier?—it can actually be a good idea if the environment is complex and building a complete universal plan is infeasible. Using a reinforcement-learning algorithm in such an environment can help the agent find appropriate behavior for the most common and important states [9, 43]. The most noteworthy example of this technique remains one of the biggest successes of reinforcement learning—Tesauro's backgammon-learning program [159], which is now reliably ranked as one of the world's best players.

1.2 Formal Models

So far, I have been very casual in discussing environments and their models. To explore methods for automatically finding optimal behavior for environments, however, I must now be very specific about the models I am concerned with and what we mean by optimal behavior.

Russell and Norvig [132] argue that any problem in artificial intelligence can be viewed as a situation faced by an agent interacting with some environment. Following their example, I list below some dimensions along which environments may vary, and where the environments I address in this thesis fall on these dimensions.

- *finite vs. continuous states*

  Is there a finite collection of states in which the environment can be, like the board positions in a game of chess, or are the states better viewed as lying in a continuum, like the position of the sun in the sky? Continuous state-space environments can be turned into finite state-space environments by discretizing the state space, although this can make them difficult to solve. I will focus on finite state spaces; however, in Chapter 7 I will show how to reason about a continuum of possible *beliefs* that result from a particular finite-state problem. I also consider a particular limiting case of finite state spaces, that of an environment with a single state. Other authors [112] have addressed environments with continuous state spaces.

- *finite vs. continuous actions*

The set of action choices can also be either finite or continuous; again I will be mainly concerned with the finite case, although Chapter 5 examines a model in which the selected actions are actually continuous probability distributions over a finite set of choices. Continuous actions can also be discretized, although specific algorithms [8] can solve this type of environment more effectively.

- *episodic vs. sequential*

  In an episodic environment, the agent faces the same problem over and over again. I am more concerned with sequential environments, in which the agent makes a sequence of interrelated decisions without necessarily being reset to a starting state. Episodic environments can be viewed as a degenerate type of sequential environment.

- *accessible vs. inaccessible*

  The agent makes its decisions on the basis of its perception of the environment. If the observations it makes are sufficient to reveal the entire state of the environment, the environment is accessible, or completely observable; otherwise it is inaccessible, or partially observable. Partially observable environments in which the perception of the agent does not change over time are unobservable. I consider both accessible and inaccessible environments in this thesis.

- *Markovian vs. non-Markovian*

  In a Markovian environment[1], the future evolution of the system can be predicted on the basis of the environment's state. In non-Markovian environments, such as the non-stationary environments mentioned earlier, it is often important to remember something about earlier states to predict the future accurately. My focus is on Markovian environments because of their relative tractability, although I also consider environments that appear non-Markovian because they are partially observable.

- *fixed vs. dynamic*

[1] Markov was a Russian mathematician whose work made use of the assumption that history is completely disregarded. Gilbert Strang, feeling this assumption provided a pessimistic view of human existence, commented [152], "Perhaps even our lives are examples of Markov processes, but I hope not." In fact, all the Markov assumption really says is that anything about a system's history that is relevant to how it will develop in the future is somehow present in the description of the current state, an assumption most physicists are willing to make about our universe.

An agent in a dynamic environment must contend with the fact that the state may change while it is deliberating on a choice of action. I focus exclusively on the simpler fixed environments, although they less accurately reflect the problems faced by real agents. This class of environments is defined by the assumption that the agent can make decisions fast enough that the state does not change between perception and action.

- *deterministic vs. stochastic*

  As mentioned earlier, it is important for algorithms to deal with the possibility that the environment contains stochastic transitions. I sometimes narrow the focus to situations in which all transitions are deterministic—there is exactly one next state for each combination of state and action—to help understand how decision making is simplified in this case.

- *synchronous vs. asynchronous*

  In the synchronous environments I consider, time advances only when the agent takes an action. Put another way, precisely one state transition in the environment occurs for each action the agent takes. In asynchronous or continuous-time models, the environment does not "wait" for the agent to take an action but instead changes continually; the actions serve as synchronizing events during which the agent and environment interact. Asynchronous environments are more general, although assuming a synchronous environment is weaker than assuming that each action takes a fixed amount of time. It is possible to approximate an asynchronous environment by a synchronous one by discretizing time [61].

- *single vs. multiple agent*

  For simplicity, I consider environments with either one or two agents. Environments with more agents are worthy of study but can be extremely complicated to analyze because of the many ways the goals and experiences of the different agents can interact.

In summary, the environments covered in this thesis share the properties of being finite state, finite action, sequential, Markovian, fixed, and synchronous, can be completely or partially observable, stochastic or deterministic, and can contain one or two agents.

	single agent	multiagent
state known	MDP	Markov game
state observed indirectly	POMDP	incomplete-information game

Table 1.1: Relationships among several Markov models.

In later chapters, I examine six related models in detail: Markov decision processes, generalized Markov decision processes, alternating Markov games, Markov games, partially observable Markov decision processes, and information-state Markov decision processes. To illustrate the relationships among these models, Table 1.1 arranges several of them into a 2×2 grid. Markov decision processes (MDPs) are fixed, stochastic environments in which a single agent issues actions given knowledge of the current state. Markov games generalize this model to allow a pair of agents to control state transitions, either jointly or in alternation. A partially observable Markov decision process (POMDP) consists of a single agent that must make decisions given only partial knowledge of its current state. In incomplete-information games, multiple agents control the transitions in the environment and the agents have incomplete and perhaps differing knowledge of the environment's state.

To date, MDPs have been the most actively studied of these models. They have received nearly 40 years of attention from the operations-research community, and are a current favorite topic among reinforcement-learning and decision-theoretic planning researchers. Markov games were developed in the economics and game-theory communities and predate even MDPs. Reinforcement-learning researchers have studied these models in some detail. POMDPs have recently been the focus of much excitement in the artificial-intelligence and machine-learning communities. Although their study dates back to the early 1960s, they have not received nearly the attention enjoyed by MDPs, in part because of their complexity. I do not address incomplete-information games; although these model are of great interest to individuals studying, for example, the coordination of multiple robots, only one group of researchers has explored algorithms for incomplete-information games [85], and only for a subset of these models.

1.3 Evaluation Criteria

The previous section began a formal treatment of environment models that I will expand upon in the coming chapters. In this section, I describe how policies, planning algorithms, and reinforcement-learning algorithms can be evaluated and compared.

1.3.1 Policies

A policy induces a probability distribution over the set of all possible sequences of states and actions for each possible initial state of the environment. This means that, in principle, it is possible to take a description of a policy, an environment, and an initial state, and compute the probability that a given sequence of states and actions will occur.

An *objective function* takes the set of possible state and action sequences and their probabilities and produces a single number, called a value. Planning and reinforcement-learning algorithms seek an optimal policy, one that maximizes the objective function, so the choice of objective function is a critical part of the statement of the problem to be solved. In this section, I present a set of objective functions that can be used to evaluate policies. Many of these objective functions have found their use in applications, others have not. My classification is intended not to exhaust cover the range of possible objective functions, but to provide a vocabulary for expressing the objective functions appearing in later chapters.

Many important objective functions can be constructed using the general framework in Table 1.2, which shows how a value can be defined as a set of transformations on the set of all possible sequences of states and actions.

1. Starting with the set of all possible state and action sequences, the objective function replaces each transition with a transition value, which is a representation of its "goodness." For an objective function measuring rewards, each transition is replaced by its reward value. For an objective function measuring steps, each transition is assigned a constant value, say -1. For an objective function sensitive only to goals, all transitions are assigned a value of zero unless the transition is to a goal state.

2. The sequence of transition values can be truncated according to the horizon;

finite-horizon criteria cut off the sequence at a prespecified length, while goal-based criteria cut off each sequence when it first enters a goal.

3. Each sequence of remaining transition values is then summarized. Undiscounted criteria compute a simple sum of transition values, discounted criteria compute a sum in which later terms are scaled down according to the discount factor β, and average-reward criteria compute an average of the transition values.[2]

4. At this point, each possible sequence of states and actions has been reduced to a single summary value. The sequence values now need to be summarized to compute a single summary value for the policy. Under expected-reward criteria, this involves taking an average of the sequence values weighted by the probability of the associated sequence. Best-case and worst-case criteria are defined by summarizing according to the largest and smallest sequence values, respectively.

5. In multiagent environments, an additional step must be introduced. It is possible that the other agents can modify the probability of various sequences of states and actions. In a cooperative environment, other agents act so as to maximize the objective function, whereas in a competitive environment, they act so as to minimize the objective function.

The elements of Table 1.2 make it possible to evaluate a fixed policy. An optimal policy is one that maximizes the value.

Two hundred forty-three objective functions can be constructed by combining one element from each category. Although all these combinations appear to be meaningful, several deserve special mention because of the attention they have received in the literature. Table 1.3 lists several combinations and a sample of references that have used them.

The algorithms in this thesis find policies that maximize or minimaximize expected discounted reward over the infinite horizon. I chose these objective functions because they have mathematical properties that make them easy to work with. In addition, all the other objective functions can be viewed as special cases of the infinite-horizon expected discounted reward objective function,[3] although specialized algorithms are almost always more efficient in these cases.

[2] In the case of infinite-horizon average reward, it is necessary to compute the average as a limit.

[3] For example, the finite-horizon criterion can be constructed by making copies of the states, and optimal average-reward policies can be found by setting the discount factor close to 1.

state/action sequences

q_1: $s_1, a_1 \rightarrow s_2, a_2 \rightarrow s_3, a_3 \rightarrow s_4, a_4 \rightarrow s_5, a_5 \rightarrow s_6, a_6 \rightarrow s_7, a_7 \rightarrow \ldots$

q_2: $s_1, a_1 \rightarrow s_2, a_2 \rightarrow s_3, a_3 \rightarrow s_4, a_4 \rightarrow s_5, a_5 \rightarrow s_6, a_6 \rightarrow s_7, a_7 \rightarrow \ldots$

q_3: $s_1, a_1 \rightarrow s_2, a_2 \rightarrow s_3, a_3 \rightarrow s_4, a_4 \rightarrow s_5, a_5 \rightarrow s_6, a_6 \rightarrow s_7, a_7 \rightarrow \ldots$

…

transition values

q_1: +0.2, +0.1, -9.0, +0.0, +7.1, +5.0, -2.2, -8.7, +3.9, …

q_2: +2.8, +4.0, +4.8, +2.9, +7.4, -2.4, -2.8, +0.4, +4.0, …

q_3: +3.3, -3.7, +5.0, +6.2, +0.0, +6.9, -3.6, -1.9, -5.7, …

…

horizon truncation

q_1: +0.2, +0.1, -9.0, +0.0, +7.1, +5.0, -2.2, -8.7, +3.9, …

q_2: +2.8, +4.0, +4.8, +2.9, +7.4, -2.4, -2.8, +0.4, +4.0, …

q_3: +3.3, -3.7, +5.0, +6.2, +0.0, +6.9, -3.6, -1.9, -5.7, …

…

sequence summary

q_1: -2.38 q_2: +11.45 q_3: 6.13 …

value summary

-16.13

Figure 1.3: Creating objective functions.

	transition values
reward	$r_t :=$ reward function for state s_t and action a_t
steps	$r_t := -1$
goals	$r_t := +1$ if s_{t+1} is a goal, $+0$ otherwise
	horizon truncation
finite	$k :=$ fixed value
infinite	$k := \infty$
goal	$k :=$ first step into goal
	sequence summary
discounted	$v(q) := \sum_{t=0}^{k} \beta^t r_t$
undiscounted	$v(q) := \sum_{t=0}^{k} r_t$
average	$v(q) := \lim_{l=0}^{k} \sum_{t=0}^{l} r_t / l$
	value summary
expected	$\sum_q v(q) \Pr(q)$
worst case	$\min_q v(q)$
best case	$\max_q v(q)$
	other agents
none	–
cooperative	maximum
competitive	minimum

Table 1.2: Choices for the components of the objective function.

maximum expected discounted reward over the infinite horizon (Chapter 2)
minimax expected discounted reward over the infinite horizon (Chapter 5)
maximum worst-case discounted reward over the infinite horizon [62]
minimax expected average reward over the infinite horizon [183]
maximum expected average reward over the infinite horizon [102]
maximum expected undiscounted reward until goal (cost-to-go) [29]
minimax expected undiscounted goal probability [36]
maximum expected undiscounted goal probability [87]
maximum multiagent discounted expected reward [22]

Table 1.3: Several popular objective functions.

Under the discounted objective, the discount factor $0 < \beta < 1$ controls how much effect future rewards have on the decisions at each moment, with small values of β emphasizing near-term gain and larger values giving significant weight to later situations. Concretely, a reward of r received t steps in the future is worth $\beta^t r$ to the agent now. Mathematically, the discount factor has the desirable property that if all immediate rewards are bounded, then the infinite sum of the discounted rewards is also bounded. From an applications perspective, the discount factor can be thought of as the probability that the agent will be allowed to continue gathering reward after the current step, or, from an economic perspective, as an inverse interest rate on reward [126].

1.3.2 Planning Algorithms

When is one planning algorithm better or worse than another? There is no unique best choice of evaluation criterion for comparing algorithms. In fact, any of the criteria from the previous section for evaluating the possible sequences of rewards gained by a policy could be used to evaluate the possible sequences of policies produced by a successive-approximation planning algorithm.

As a first cut, algorithm A is better than algorithm B if A can find an optimal policy and B cannot, or if A is more likely than B to find an optimal policy. I am mainly interested in algorithms that are guaranteed to find an optimal policy, although sometimes it is important to tolerate a fixed margin of error.

When both algorithms can find optimal policies, the better algorithm is the one that can do so more quickly. Of course, not all algorithms perform well on all problems and some algorithms use randomization, so once again it is important to decide how to summarize the run time of algorithms over a class of environments. As in the previous section, algorithms can be summarized according their average performance over all environments or on one that makes them perform the worst, and by average or worst-case run times.

In this thesis, an algorithm's run time is measured by its worst-case expected run time; that is, the average run time of the algorithm on the worst possible problem instance. This measure is very popular in theoretical computer science because it captures the reality that the algorithm has no control over the problem instance it must solve.

More concretely, the *complexity* of a planning algorithm is its worst-case average run

intractable	EXPTIME-complete
presumed intractable	PSPACE-complete NP-complete
presumed tractable	NP∩co-NP
tractable	RP P NC

Figure 1.4: Some complexity classes.

time expressed as a function of the size of a description of the environment, primarily the number of transition probabilities and rewards. For most of the problems in this thesis, I use a very coarse measure of complexity that is relatively insensitive to the details of how environments are described. For present purposes, an algorithm is efficient if its complexity can be bounded above by some polynomial function, for example n^5 where n is the size of the environment description. If an exponential function is a lower bound on the complexity of an algorithm, the algorithm is inefficient.

For many problems, it can be shown that it is unlikely that any efficient algorithm exists by appealing to results from complexity theory, which classifies problems by their difficulty. Figure 1.4 summarizes some of the most important classes used in this thesis. Of these, the problems in RP, P, and NC are the only ones known to be solvable in polynomial time. For the other classes, the best known algorithms take exponential time. However, the range of difficulty among these classes is large: problems in NP∩co-NP might be solvable in polynomial time by the best algorithm; NP-complete problems can often be approximated in polynomial time; PSPACE-complete problems are often quite difficult to approximate; and EXPTIME-complete problems are known not to be solvable in polynomial time in the worst case.

Much of the uncertainty here derives from the famous open problem "Does P equal NP?" If P=NP, then all the classes in Figure 1.4 from NP-complete down to NC have efficient algorithms. If P≠NP, then the classes from NP-complete up to EXPTIME do not have efficient algorithms. Additional background information on complexity classes can be found in Papadimitriou's textbook [115] and the summary in Section A.1.

1.3.3 Reinforcement-learning Algorithms

In a reinforcement-learning scenario, the agent must solve the same basic problem faced in planning, but must do so without a correct description of the environment. As a result, the range of choices for evaluating reinforcement-learning algorithms are quite large; it is reasonable to measure the complexity of the per-experience computation, the number of experiences needed to identify a near-optimal policy, and the value attained by a policy learned after a particular number of experiences. A truly optimal reinforcement-learning agent would take actions to maximize the objective function under the restriction that its knowledge of the environment is incomplete. Such an approach can be formalized [108], but is believed to be intractable except in the simplest case in which there is only one state: see Gittins' groundbreaking work on this topic [56].

In the interest of simplicity, other measures are typically used to compare reinforcement-learning algorithms. One approach is to measure the *regret*, the amount of additional reward an algorithm would have received had it behaved according to the optimal policy all along. This model seems quite appropriate for evaluating reinforcement learning, but has received little formal attention.

An even simpler approach is to compare reinforcement-learning algorithms by ignoring the learning phase altogether and asking whether an agent will, in the limit, discover optimal behavior. This is the best-studied approach and the one I use here. It basically categorizes reinforcement-learning algorithms as "good" if they can be used to produce a sequence of policies that become arbitrarily close to optimal, and "bad" otherwise. While this is an important and useful way to categorize algorithms, a better categorization would be based on how quickly algorithms identify near-optimal policies; because such *convergence-rate analyses* can be quite complex, very little theoretical work has been done in this direction.

1.4 Thesis Summary

A main concern of researchers in artificial intelligence is how to formalize real-world problems. This translates to the search for models that trade off two conflicting forces: models should be simple enough to permit efficient computer manipulation and complex enough to capture the relevant aspects of the real world. Although I do not propose any novel models of real-world problems, the new results I present extend the understanding of several existing models. In particular, I show how several existing models can be manipulated more easily, analyzed more precisely, and viewed more generally than was possible previously.

Each chapter of the thesis examines a particular model in detail. They are, by chapter,

2. **Markov decision processes**

   Completely observable, single-agent environments. The most basic sequential decision-making problem. This chapter provides much of the conceptual framework for the remainder of the thesis. A new result in this chapter is an analysis of the complexity of the policy-iteration algorithm.

3. **Generalized Markov decision processes**

   A set of models with a common mathematical framework. Includes all the other models discussed in the thesis. This model is new, as is the convergence proof for reinforcement learning in this model. The chapter itself is very mathematically oriented.

4. **Alternating Markov games**

   Completely observable, two-agent zero-sum environments. Each agent takes some number of actions and then turns control over to the other agent. I present the first convergence proof for reinforcement learning in games.

5. **Markov games**

   Alternating games in which agents take actions simultaneously. I present a method for finding optimal stochastic policies in this class of games.

6. **Partially observable Markov decision processes**

   Partially observable, single-agent environments. Very difficult to solve. The focus in this chapter is on complexity results and I give the first proof of intractability for this class of models.

7. **Information-state Markov decision processes**

   A type of continuous state-space Markov decision process derived from partially observable environments. I present a new algorithm for solving finite-horizon problems, along with theoretical results that show it is the most efficient algorithm of its kind.

To help highlight the similarities and differences among the algorithms for solving these models, each chapter follows the same organization:

1. **Introduction**

   Describes the relevant aspects of the model, how it is similar to and different from other models, and its relation to real-world problems.

2. **The model**

   The formal definition of the model, what it means to act optimally in the model, and foundational mathematical results.

3. **Algorithms for solving the model**

   Describes the major approaches to solving the model along with enough analysis to clarify the basic structure of the approaches.

4. **Algorithmic analysis**

   Analyses of the computational complexity of the algorithms.

5. **Complexity results**

   Analyses of the difficulty of finding optimal policies in the model and its variations.

6. **Reinforcement learning in the model**

   Description of relevant reinforcement-learning algorithms and convergence results.

7. **Open problems**

   List of outstanding unresolved questions pertaining to planning and reinforcement learning in the model.

8. **Related work**

   Summary of relevant contributions from other researchers.

9. **Contributions**

   Summary of the new results presented in the chapter.

In addition, each chapter has an associated appendix with important results whose formal arguments are too complex or otherwise unenlightening to be included in the main text.

1.5 Additional Remarks

The focus of this thesis is on algorithms, and there are important mathematical details that are left unproven. Wherever possible, I refer to other sources for additional mathematical background. Similarly, a great deal of relevant information is left unsaid regarding the efficient implementation of these algorithms on existing computer systems using existing computer languages. In nearly all cases, I provide "pseudocode" implementations of the important algorithms; that is, I describe the algorithms in a computer-language-like notation that, with some work, could be made to run on a computer. The conventions I use for describing algorithms are explained in Section A.2.

The mathematical sophistication assumed is roughly at the level of basic analysis. However, even the most mathematical of the results can be appreciated without this background. The rest of the material should be accessible to a typical computer scientist. One possible exception is that I assume a familiarity with linear programming. A brief overview of this topic is given in Section A.3.

I present example environments throughout in a graphical notation of which Figure 1.5 is a simple example. The circles are the states of the environment, with their "names" written inside. Each arrow that leaves a state is an action and is labeled with its name. The numerical label on the action is the immediate reward; I write immediate reward values with a leading sign to help distinguish them from transition probabilities. Arrows sometimes split; this represents a stochastic transition, with the individual transitions labeled with their probability. Thus in Figure 1.5, action a_2 from state s_1 results in an immediate reward of $+1$ and a transition back to state s_1 with probability 0.2.

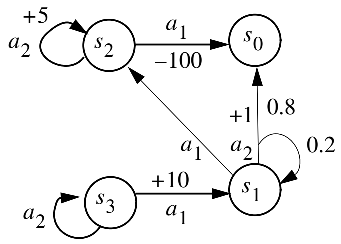

Figure 1.5: An illustration of a simple environment.

1.6 Related Work

The baseball example was inspired by a passage in a book by pitcher Tom Seaver [142]; the hypothetical batter is Babe Ruth. The network-monitoring example was inspired by conversations with my colleagues at Bellcore; a simple study of reinforcement learning in a different telecommunications domain was undertaken by Boyan and Littman [27]. Additional applications are described in Puterman's textbook [126].

Kaelbling's book [72] provides a philosophical discussion of agents, along with an exploration of several kinds of sequential decision-making problems. Russell and Norvig's introductory textbook [132] views all of artificial intelligence from an agent perspective.

Interrelationships between the undiscounted-reward and the discounted-reward criteria were discussed by Blackwell [19]. A survey of results concerning the average-reward criterion was written by Arapostathis et al. [4]. Fernández-Gaucherand, Ghosh and Marcus [52] explored combinations of discounted and average reward as a way to better trade off short-term and asymptotic reward. In the area of reinforcement-learning, the average-reward criterion was studied first by Schwartz [141] and later in detail by Mahadevan [102].

1.7 Contributions

This chapter introduced the core concepts used in the remainder of the thesis: agents, environments, rewards, policies, planning and learning. I showed where the problems I address fall on a categorization of environments inspired by a similar categorization described by Russell and Norvig [132]. I explained how algorithms and policies are evaluated in my work, and presented a novel categorization of objective functions that includes those used in my work as well as those used by many other researchers.

Chapter 2

Markov Decision Processes

Portions of this chapter have appeared in earlier papers: "Planning and acting in partially observable stochastic domains" [73] with Kaelbling and Cassandra, "An introduction to reinforcement learning" [74] with Kaelbling and Moore, and "On the complexity of solving Markov decision problems" [96] with Dean and Kaelbling.

Consider the problem of creating a policy to guide a robot through an office building. The robot's actions take it from hallway intersection to intersection, but are not completely reliable. Sometimes an action fails, overshoots, or results in the robot turning too far. Fortunately, perfect sensors allow the robot to perceive the effects of its error-plagued actions.

The assumption of perfect sensors is central to this chapter. Because the agent can directly perceive all aspects of its current state that might be necessary to predict the probability of the next state given its action, it is not necessary for the agent to retain any history of its past actions or states to make optimal decisions.

2.1 Introduction

In this chapter I address the problem of choosing optimal actions in completely observable stochastic domains. The robot problem described above can be modeled as a *Markov decision process* (MDP), as can other problems in robot navigation, factory process control, transportation logistics, and a variety of other complex real-world situations.

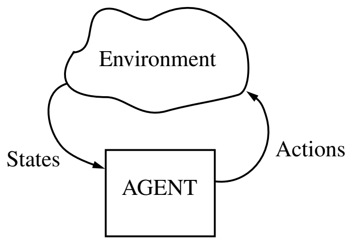

Figure 2.1: An MDP models the synchronous interaction between agent and environment.

The problem addressed is, given a complete and correct model of the environment dynamics and a goal structure, find an optimal way to behave. Versions of this problem have been addressed in the artificial-intelligence literature as planning problems, where the focus is on goal-oriented problems in large, deterministic domains. Because we are interested in stochastic domains, we must depart from the traditional model and compute solutions in the form of policies instead of action sequences.

Much of the content of this chapter is a recapitulation of work in the operations-research literature [126, 15, 44, 46, 68, 13] and the reinforcement-learning literature [153, 173, 10, 145]. The concepts and background introduced here will be built upon in all the succeeding chapters.

2.2 Markov Decision Processes

Markov decision processes are the simplest family of models I will consider. Later chapters depend on the results and concepts introduced here to address more complex generalizations of MDPs. An MDP is a model of an agent interacting synchronously with its environment. As shown in Figure 2.1, the agent takes as input the state of the environment and generates as output actions, which themselves affect the state of the environment. In the MDP framework, it is assumed that, although there may be a great deal of uncertainty about the effects of an agent's actions, there is never any uncertainty about the agent's current state—it has complete and perfect perceptual abilities.

2.2.1 Basic Framework

A Markov decision process is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \beta \rangle$, where

- $\mathcal{S}$ is a finite set of states of the environment;
- $\mathcal{A}$ is a finite set of actions;
- $T : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is the *state transition function*, giving for each state and agent action, a probability distribution over states ($T(s, a, s')$ is the probability of ending in state s', given that the agent starts in state s and takes action a);
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, giving the expected immediate reward gained by the agent for taking each action in each state ($R(s, a)$ is the expected reward for taking action a in state s); and
- $0 < \beta < 1$ is a discount factor.

In this chapter, I consider MDPs with finite state and action spaces. Many of the important results apply to infinite-state MDPs as well. Chapter 3 examines these results from a more general perspective.

2.2.2 Acting Optimally

Agents should act in such a way as to maximize some measure of the long-run reward received. Section 1.3.1 presented several objective functions, including discounted finite-horizon optimality, average-reward optimality, and discounted infinite-horizon optimality. I focus on algorithms for the discounted infinite-horizon case with some attention to the simpler finite-horizon case. Most of the results for infinite-horizon discounted MDPs apply to undiscounted problems in which agents are guaranteed to reach a zero-reward absorbing state regardless of policy, the all-policies-proper case (see Chapter 3 for more details).

A policy is a description of the behavior of an agent. I consider two important kinds of policies: stationary and non-stationary. A *stationary policy*, $\pi : \mathcal{S} \to \mathcal{A}$, specifies, for each state, an action to be taken. The choice of action depends only on the state and is independent of the time step. A *non-stationary policy* is a sequence of state-action mappings, indexed by time. In the non-stationary policy $\delta = \langle \pi_t, \ldots, \pi_1 \rangle$, the mapping π_t is used to choose the action on the tth-to-last step as a function of the current state. In the finite-horizon model, there is rarely a stationary optimal policy—the way an agent chooses its actions on the final step is generally different than the way it chooses them when it has a large number of steps left. In the discounted infinite-horizon model, the quantity $1-\beta$ can be viewed as the probability that the agent will cease to accrue additional reward; therefore, it is as if the agent always has a constant expected number of steps remaining: $1/(1-\beta)$. Because the expected distance to the horizon never changes, there is no reason to change action strategies as a function of time—there is a stationary optimal policy [15].

Given a policy, we can evaluate the expected long-run value an agent could expect to gain from executing it. In the finite-horizon case, let $\delta = \langle \pi_k, \ldots, \pi_1 \rangle$ be a k-step non-stationary policy and let $V_t^\delta(s)$ be the expected future reward starting in state s and executing non-stationary policy δ for t steps. The value of the final step is the immediate reward $V_1^{\delta_t}(s) = R(s, \pi_1(s))$. For $t > 1$, we can define $V_t^{\delta_t}(s)$ inductively by

$$V_t^{\delta_t}(s) = R(s, \pi_t(s)) + \beta \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_{t-1}^{\delta_t}(s').$$

The t-step value of being in state s and executing non-stationary policy δ is the immediate reward, $R(s, \pi_t(s))$, plus the discounted expected value of the remaining $t-1$ steps. To evaluate the remaining steps, we consider all possible resulting states s', the likelihood of their occurrence $T(s, a, s')$, and their $(t-1)$-step value under policy δ, $V_{t-1}^\delta(s')$.

Let $V^\pi(s)$ be the expected discounted future reward for starting in state s and executing stationary policy π indefinitely. The infinite-horizon value is recursively defined by

$$V^\pi(s) = R(s, \pi(s)) + \beta \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V^\pi(s').$$

The value function for policy π is the unique solution of this set of simultaneous linear equations, one for each state s. A subroutine for finding the infinite-horizon value function for a given policy appears in Table 2.1; it will be used later in more complex algorithms. The system of linear equations can be solved by Gaussian elimination or any of a number of other methods [40].

Now we know how to compute a value function, given a policy. We can also define a policy based on a value function. Given any value function V, the *greedy policy* with respect to that value function, π_V, is defined as

$$\pi_V(s) = \operatorname*{argmax}_a \left[R(s, a) + \beta \sum_{s' \in \mathcal{S}} T(s, a, s') V(s') \right].$$

```
evalMDP(π, ⟨S, A, T, R, β⟩) := {
    Solve the following system of linear equations:
        find: v[s]
        s.t.: v[s] = R(s, π(s)) + β Σ_{s'∈S} T(s, π(s), s')v[s'], for all s ∈ S
    return v
}
```

Table 2.1: Computing the value function for a given policy.

This policy is obtained by taking the action in each state with the best one-step value according to V.

It is not hard to find the optimal finite-horizon policy for a given MDP, $\delta_k^* = \langle \pi_k^*, \ldots, \pi_1^* \rangle$. For $1 \le t \le k$, we can define π_t^* as follows. On the final step, the agent should maximize its immediate reward,

$$\pi_1^*(s) = \operatorname*{argmax}_a R(s, a).$$

We can define π_t^* in terms of the value function for the optimal $(t-1)$-step policy $V_{t-1}^{\delta_k^*}$ (written for simplicity as V_{t-1}):

$$\pi_t^*(s) = \operatorname*{argmax}_a \left[R(s, a) + \beta \sum_{s' \in \mathcal{S}} T(s, a, s') V_{t-1}(s') \right],$$

it need not be unique.

In the discounted infinite-horizon case, given an initial state s, we want to execute the policy π that maximizes $V^\pi(s)$. Howard [68] showed that there exists a stationary policy π^* that is optimal for every starting state. The value function for this policy, written V^*, is defined by the set of equations

$$V^*(s) = \max_a \left[R(s, a) + \beta \sum_{s' \in \mathcal{S}} T(s, a, s') V^*(s') \right], \tag{2.1}$$

and any greedy policy with respect to this value function is optimal [126].

The presence of the maximization operator in Equation 2.1 means the system of equations is not linear—Gaussian elimination is not sufficient to solve it. In the next section, I explore algorithms that solve the MDP problem; they find an optimal policy and value function for an MDP, given its description in terms of T, R, and β.

```
ValueIterationMDP(⟨S, A, T, R, β⟩, ε) := {
    foreach s ∈ S V_0(s) := 0
    t := 0
    loop
        t := t + 1
        foreach s ∈ S {
            foreach a ∈ A
                Q_t(s, a) := R(s, a) + β Σ_{s'∈S} T(s, a, s')V_{t-1}(s')
            π_t(s) := argmax_a Q_t(s, a)
            V_t(s) := Q_t(s, π_t(s))
        }
    until max_s |V_t(s) − V_{t-1}(s)| < ε
    return π_t
}
```

Table 2.2: The value-iteration algorithm for MDPs.

2.3 Algorithms for Solving Markov Decision Processes

There are many methods for finding optimal policies for MDPs. In this section, I describe several of the more basic approaches. All of these can be found in Puterman's textbook [126].

2.3.1 Value Iteration

Value iteration proceeds by computing the sequence V_t of discounted finite-horizon optimal value functions, as shown in Table 2.2. It makes use of an auxiliary function, $Q_t(s, a)$, which is the t-step value of starting in state s, taking action a, then continuing with the optimal $(t-1)$-step non-stationary policy; these state-action value functions are also known as *Q functions*. The algorithm terminates when the maximum difference between two successive value functions, the *Bellman error magnitude*, is less than some predetermined ϵ.

It can be shown that there exists a t^*, polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, $\log \max_{s,a} |R(s,a)|$, and $1/(1-\beta)$, such that the greedy policy with respect to V_{t^*} is an optimal infinite-horizon policy [162]. Rather than calculating a bound on t^* in advance and running value iteration for that long, we can use the Bellman error magnitude to decide when our current value function is good enough to generate a near-optimal greedy policy. To state this precisely, if $|V_t(s) - V_{t-1}(s)| < \epsilon$ for all s, then the value of the greedy policy with respect to V_t does not differ from V^* by more than $2\epsilon\beta/(1-\beta)$ at any state. That is,

$$\max_{s\in\mathcal{S}} |V^{\pi_{V_t}}(s) - V^*(s)| < 2\epsilon\frac{\beta}{1-\beta}.$$

This result [180] is discussed in more detail in Section 3.3.2. Tighter bounds may often be obtained using the *span semi-norm* on the value function [126].

2.3.2 Policy Iteration

In the version of value iteration discussed in the previous section, each value function V_t can be interpreted as an *approximation* of the value function of the optimal infinite-horizon policy π^*, or as the value function for the *optimal* non-stationary policy for a t-step discounted finite-horizon MDP. This second interpretation comes from the fact that the initial value function, V_0, is defined to be zero for all states.

A different interpretation is possible when we define V_1 as follows: let π_0 be the greedy policy for the zero value function, V_0, and let $V_1 = V^{\pi_0}$, the value function for policy π_0. Define a sequence of infinite-horizon non-stationary policies where $\delta_t = \langle \pi_t, \ldots, \pi_1, \pi_0, \pi_0, \pi_0, \ldots \rangle$, that is, δ_t is the infinite-horizon policy that follows π_t, then π_{t-1}, and so on down to π_0, which is repeated indefinitely. We can view V_t, the value function obtained on the tth iteration starting from V_1, as the infinite-horizon value function corresponding to δ_t.

This revised value-iteration algorithm shares the convergence properties attributed to the algorithm in Table 2.2 (see Section 3.3.1), and possesses a few special properties of its own. Since V_t is the value function for the non-stationary infinite-horizon policy δ_t, and V^* is the optimal infinite-horizon policy, then $V^*(s) \geq V_t(s)$ for all s. This is because no infinite-horizon policy, stationary or not, can have a higher expected discounted reward from any state than that given by the optimal value function [15]. As a result, V_t converges to V^* from below.

A more useful fact is that by adopting the state-action mapping π_t as a stationary infinite-horizon policy, an agent is guaranteed total expected reward that is no worse than it would obtain following δ_t [126]. That is, $V^{\pi_t}(s) \geq V^t(s)$, for all $s \in \mathcal{S}$.

Combining these insights leads to an elegant approach to solving MDPs, due to Howard [68]. Like value iteration, *policy iteration* (Table 2.3) computes a sequence of value functions, V_t. In policy iteration, however, each value function V_t is the value

```
PolicyIterationMDP(M = ⟨S, A, T, R, β⟩) := {
    foreach s ∈ S V_0(s) := 0
    t := 0
    loop
        t := t + 1
        foreach s ∈ S and a ∈ A {
            Q_t(s, a) := R(s, a) + β Σ_{s'∈S} T(s, a, s') V_{t−1}(s')
            π_t(s) := argmax_a Q_t(s, a)
            V_t := evalMDP(π_t, M)
        }
    until V_{t−1}(s) = V_t(s) for all s
    return π_t
}
```

Table 2.3: The policy-iteration algorithm for MDPs.

function for the greedy policy with respect to the previous value function V_{t-1}.

Puterman [126] shows that the sequence of value functions produced in policy iteration converges to V^* no more slowly than the value functions produced in value iteration; this result is discussed in some detail in Section B.1. Policy iteration can converge in fewer iterations than value iteration; however, the increased speed of convergence of policy iteration can be more than offset by the increased computation per iteration.

Another fundamental difference between value iteration and policy iteration is the stopping criterion. Whereas value iteration can converge to the optimal value function very gradually, policy iteration proceeds in discrete jumps. In particular, each value function generated in policy iteration is associated with a particular policy (of which there are $|\mathcal{A}|^{|\mathcal{S}|}$), and each value function V_t is strictly closer to V^* than is V_{t-1} [126]. Putting these facts together tells us that policy iteration requires at most $|\mathcal{A}|^{|\mathcal{S}|}$ iterations to generate an optimal value function.

2.3.3 Linear Programming

We saw earlier that the infinite-horizon value function for a policy π can be expressed as the solution to a system of simultaneous linear equations. Table 2.1 shows how this mathematical equivalence can be exploited to derive an efficient algorithm for computing V^π.

```
mdpLP(⟨S, A, T, R, β⟩) := {
    Solve the following linear program:
        minimize: Σ_s v[s]
        s.t.: v[s] ≥ R(s, a) + β Σ_{s'∈S} T(s, a, s')v[s'], for all s ∈ S and a ∈ A
        variables: v[s] for all s ∈ S
    return v
}
```

Table 2.4: Solving an MDP via linear programming.

We also saw that the optimal value function V^* can be expressed as the solution to a system of simultaneous equations, given in Equation 2.1. Unfortunately, the maximization operators in these equations render them non-linear, so nothing as simple as Gaussian elimination will suffice to solve them.

The V^* equations can be expressed in the more descriptive mathematical language of *linear programming*. A linear program consists of a set variables, a set of linear inequalities over these variables, and a linear objective function. D'Epenoux [45] showed how to solve an MDP by expressing the system of equations defining the optimal value function as a linear program (Table 2.4).

The intuition here is that, for each state s, the optimal value from s is no less than what would be achieved by first taking action a, for each $a \in \mathcal{A}$. The minimization ensures that we choose the *least* upper bound (the maximum, in other words) for each of the $v[s]$ variables.

An important fact from the theory of linear programming is that every linear program has an equivalent linear program in which the roles of the variables and the constraints are reversed. The resulting linear program, known as the *dual*, can also be used to solve MDPs. One advantage of the dual formulation is that it makes it possible to express and incorporate additional constraints on the form of the policy found [88].

The dual linear program appears in Table 2.5. The $f[s', a]$ variables can be thought of as indicating the amount of "policy flow" through state s' that exits via action a. Under this interpretation, the constraints are flow-conservation constraints that say that the total flow exiting state s' is equal to the flow beginning at state s' (always 1) plus the flow entering state s' via all possible combinations of states and actions weighted by their probability. The objective, then, is to maximize the value of the

mdpDUAL($\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \beta \rangle$) := {
Solve the following linear program:
maximize: $\sum_a \sum_s R(s,a) f[s,a]$
s.t.: $\sum_{a \in \mathcal{A}} f[s', a] = 1 + \beta \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} T(s,a,s') f[s,a]$, for all $s' \in \mathcal{S}$
variables: $f[s,a]$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
foreach $s \in \mathcal{S}$ $\pi(s) := \text{argmax}_a f[s,a]$
return evalMDP($\pi, \mathcal{M}$)
}

Table 2.5: Solving an MDP via the linear programming dual.

flow.

If the $f[s,a]$ variables constitute a feasible solution to the dual (i.e., they jointly satisfy the constraints), then $\sum_s \sum_a R(s,a) f[s,a]$ can be interpreted as the sum of the state values of the stationary *stochastic* policy that chooses action a in state s with probability $f[s,a] / \sum_{a \in \mathcal{A}} f[s,a]$. The stochastic policy corresponding to the optimal value of the $f[s,a]$ variables is optimal, as is any deterministic policy that chooses action a in state s where $f[s,a] > 0$.

2.3.4 Other Methods

There are many other methods for solving MDPs, including methods that accelerate the convergence of value iteration by keeping explicit suboptimality bounds [15] and by grouping and regrouping states throughout the process [17].

A different approach is illustrated in *modified policy iteration* [127], which has the basic form of policy iteration with the difference that a successive-approximation algorithm (basically value iteration with the policy held fixed) is used to find an approximation to the value function for policy π_t. The connection between value iteration for MDPs and successive approximation for evaluating stationary policies is explored in somewhat more detail in Chapter 3.

2.3.5 Algorithms for Deterministic MDPs

In this section I show that deterministic MDPs can be solved very efficiently by presenting a novel formulation that leads to a fast algorithm.

$$
\begin{array}{l}
\texttt{determMDP}(\langle \mathcal{S}, \mathcal{A}, N, R, \beta \rangle) := \{ \\
\quad \textbf{foreach } s \in \mathcal{S} \ \{ \\
\quad\quad \textbf{foreach } s' \in \mathcal{S} \ A^1[s, s'] := -\infty \\
\quad\quad \textbf{foreach } a \in \mathcal{A} \\
\quad\quad\quad A^1[s, N(s,a)] := \max(A^1[s, N(s,a)], R(s,a)) \\
\quad \} \\
\quad \textbf{foreach } k \in [2 \ldots |\mathcal{S}|] \ \{ \\
\quad\quad \textbf{foreach } s \in \mathcal{S} \ \{ \\
\quad\quad\quad \textbf{foreach } s'' \in \mathcal{S} \\
\quad\quad\quad\quad\quad A^k[s, s''] := \max_{s' \in \mathcal{S}}(A^1[s, s'] + \beta A^{k-1}[s', s'']) \\
\quad\quad \} \\
\quad \} \\
\quad \textbf{foreach } s \in \mathcal{S} \\
\quad\quad V[s] := \max_{s'' \in \mathcal{S}, k \in [0 \ldots |\mathcal{S}|], l \in [0 \ldots |\mathcal{S}|]} A^k[s, s''] + \beta^{k+1} A^l[s'', s''] / (1 + \beta^l) \\
\quad \textbf{return } V \\
\}
\end{array}
$$

Table 2.6: Computing the value function for a deterministic MDP.

A deterministic MDP is one in which $T(s, a, s')$ is either 0 or 1 for all $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$. The notation $N(s, a)$ represents the unique next state resulting from taking action a from state s in a deterministic MDP. In this section, I present two algorithms for this problem; one runs efficiently in parallel, and the other links the deterministic MDP problem to a general class of shortest-path problems, which results in an efficient sequential algorithm.

Papadimitriou and Tsitsiklis [116] give a dynamic-programming algorithm for solving deterministic MDPs efficiently on a parallel machine. A sequential version of their algorithm (given in Table 2.6) runs in $|\mathcal{S}|^2 + |\mathcal{S}||\mathcal{A}| + 2|\mathcal{S}|^4$ time. Here $A^k[s, s'']$ represents the maximum reward on any path of length k from state s to state s''. The value for state s can then be expressed as the maximum value path from s to s'', and then from s'' back to itself indefinitely.

Deterministic MDPs can be cast in the *closed semiring* framework, and then solved in polynomial time using a generic algorithm for solving closed semiring problems [40]. In Section B.3, I define the deterministic MDP closed semiring and prove that it has the necessary properties.

Cormen, Leiserson and Rivest [40] present a generic algorithm for solving path problems on closed semirings. The algorithm can find the optimal value function for a deterministic MDP in $|\mathcal{S}|^2 + |\mathcal{S}||\mathcal{A}| + |\mathcal{S}|^3$ time.

There are a few things to note about this new closed-semiring-based algorithm. First, its sequential run time is an improvement over the algorithm given by Papadimitriou and Tsitsiklis, which was designed to prove that the problem is in NC. Second, casting the problem in a more general framework helps highlight the similarities between deterministic MDPs and general path-related problems. Third, any advances in the area of algorithms for closed semirings immediately translate into advances for deterministic MDPs.

2.4 Algorithmic Analysis

This section provides complexity analyses of the algorithms described in the previous section. For several of the bounds, it is necessary to assume that the components of the reward and transition matrices are represented by rational numbers. We use B to designate the maximum number of bits needed to represent any numerator or denominator of β or one of the components of T or R. Although none of the individual results in this section are novel, they are presented in a way that highlights the connections between linear programming, value iteration, and policy iteration, which allows me to draw some novel conclusions concerning their complexity.

2.4.1 Linear Programming

The linear program in Table 2.4 has $|\mathcal{S}||\mathcal{A}|$ constraints and $|\mathcal{S}|$ variables and the dual given in Table 2.5 has $|\mathcal{S}|$ constraints and $|\mathcal{S}||\mathcal{A}|$ variables. In both linear programs, the coefficients have a number of bits polynomial in B. There are algorithms for solving rational linear programs that take time polynomial in the number of variables and constraints as well as the number of bits used to represent the coefficients [78, 79]. Thus, MDPs can be solved in time polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, and B. Descendants of Karmarkar's algorithm [78] are considered among the most practically efficient linear-programming algorithms.

It is popular to solve linear programs by variations of Dantzig's simplex method [41], which works by choosing a subset of constraints to satisfy with equality and solving the resulting linear equations for the values of the variables. The algorithm proceeds by iteratively swapping constraints in and out of the selected subset, continually improving the value of the objective function. When no swaps can be made to improve the objective function, the optimal solution has been found. Simplex methods differ as to their choice of *pivot rule*, the rule for choosing which constraints to swap in and out at each iteration.

Although simplex methods seem to perform well in practice, Klee and Minty [81] showed that one of Dantzig's choices of pivoting rule could lead the simplex algorithm to take an exponential number of iterations on some linear programs. Since then, other pivoting rules have been suggested and almost all have been shown to result in exponential run times in the worst case; none has been shown to result in a polynomial-time implementation of simplex. Note that these results may not apply to the use of linear programming to solve MDPs: the set of linear programs resulting from MDPs might not include the counterexample linear programs. Some progress has been made speeding up simplex-based methods, for instance, through the introduction of randomized versions of pivoting rules [20], some of which have been shown to result in subexponential complexity [75].

The fact that the optimal value function for an MDP can be expressed as the solution to a polynomial-size linear program has several important implications. First of all, it provides a theoretically efficient way of solving MDPs. Secondly, it provides a practical method for solving MDPs using commercial-grade implementations. Thirdly, it puts a convenient bound on the complexity of the optimal value function, as we will see in a moment.

Theorem 2.1 *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \beta \rangle$ be an* MDP *with transitions, rewards, and the discount factor expressed as rational numbers with numerator and denominator needing no more B bits. The components of the optimal value function for $\mathcal{M}$ are rational numbers with numerator and denominator needing no more B^* bits, and B^* is bounded by a polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, and B.*

Proof: It is well known [140, 162] that the solution to a rational linear program in which each numerator and denominator is represented using no more than B bits, can itself be written using rational numbers. The value of each variable in the solution can be represented in B^* bits where B^* is polynomial in the size of the linear program and B. The argument is based on a use of Cramer's rule to get a closed form for the solution of a system of linear equations in terms of determinants. Bounding the size of the determinant is not hard and gives a bound on the size of the components of the optimal solution.

Because the optimal value function for $\mathcal{M}$ is the solution to just such a linear program, B^* also bounds the number of bits needed to express the optimal value function for $\mathcal{M}$. $\square$

More on the relationship between linear programming and MDPs appears in Section 2.4.3.

2.4.2 Value Iteration

In this section, we analyze the value-iteration algorithm described in Section 2.3.1. When the discount factor is less than one, the sequence of value functions produced in the course of value iteration converges to the optimal value function [126].

In the general case, each iteration takes $|\mathcal{A}||\mathcal{S}|^2$ steps. The focus of this section is on the number of iterations required to reach an optimal policy. Drawing from earlier work, I will sketch an upper bound on the number of iterations and show that there are MDPs that take nearly that long to solve. More detailed discussions can be found in papers by Tseng [162] (upper bound) and Condon [37] (lower bound).

Lemma 2.1 *The number of iterations required by value iteration to reach an optimal policy is bounded above by a polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, B, and $1/(1-\beta)$.*

Proof: The proof consists of 5 basic steps.

1. *Bound the difference between the initial value function and the optimal value function, over all states.*

   Let $M = \max_{s\in\mathcal{S},a\in\mathcal{A}} |R(s,a)|$, the magnitude of the largest immediate reward. Since $\sum_{t=0}^{\infty}\beta^t M = M/(1-\beta)$, the value function for any policy will have components in the range $[-M/(1-\beta), M/(1-\beta)]$. Thus, any choice of initial value function with components in this range cannot differ from the optimal value function by more than $2M/(1-\beta)$ at any state.

2. *Show that each iteration results in an improvement of a factor of at least β in the distance between the current and optimal value functions.*

   This is the standard "contraction mapping" result for MDPs [126] that explains why the value functions in value iteration converge. It is proven in a general form in Chapter 3, though the proof for the MDP case is somewhat simpler.

3. *Give an expression for the difference between current and optimal value functions after t iterations. Show how this gives a bound on the number of iterations required for an ϵ-optimal policy.*

   After t iterations, the current value function can differ from the optimal value function by no more than $2M\beta^t/(1-\beta)$ at any state. Solving for t and using the result relating the Bellman error magnitude to the value of the associated greedy policy (Sections 2.3.1 and 3.3.2), we can express the maximum number of iterations needed to find an ϵ-optimal policy, i.e., one that has an expected value within ϵ of the optimal policy, as

   $$t^* \leq \frac{B + \log(1/\epsilon) + \log(1/(1-\beta)) + 1}{1-\beta}. \tag{2.2}$$

4. *Argue that there is a value for $\epsilon > 0$ for which an ϵ-optimal policy is, in fact, optimal.*

   Theorem 2.1 says that each component of the optimal value function for an MDP can be written using B^* bits, where B^* is polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$ and B. This means that if we can find a policy that is $\epsilon = (1/2^{B^*+1})$-optimal, the policy must be optimal.

5. *Substitute the value of ϵ from step 4 into the bound in step 3 to get a bound on the number of iterations needed for an exact answer.*

   Substituting for ϵ in Equation 2.2 reveals that running value iteration for a number of iterations polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, B, and $1/(1-\beta)$ guarantees an optimal policy.

□

This analysis shows that value iteration runs in pseudopolynomial time, and, for fixed discount factor β, it runs in polynomial time. Next, I examine a lower bound that indicates that number of iterations can grow linearly with $1/(1-\beta)\log(1/(1-\beta))$, showing that value iteration is not a polynomial-time algorithm in general.

Lemma 2.2 *Determining an optimal infinite-horizon policy via value iteration takes a number of iterations proportional to $1/(1-\beta)\log(1/(1-\beta))$ in the worst case.*

Proof: Figure 2.2 illustrates a family of deterministic MDPs, each of which consists of 3 states, labeled s_0, s_1, and s_2. From state s_0, action a_1 causes a transition to state s_1

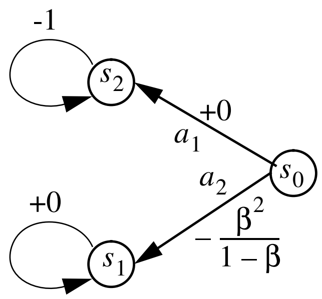

Figure 2.2: Value iteration requires number of iterations proportional to $1/(1-\beta)\log(1/(1-\beta))$ to generate an optimal solution for this family of MDPs.

and action a_2 causes a transition to state s_2. Action a_1 has no immediate reward but once in state s_1, there is a reward of -1 for every timestep thereafter. Action a_2 has an immediate reward of $-\beta^2/(1-\beta)$ but state s_2 is a zero-cost absorbing state.[1]

The discounted infinite-horizon value of choosing action a_1 from state s_0 is $-\beta/(1-\beta)$ whereas the value for action a_2 is $-\beta^2/(1-\beta)$ (larger, since $0 < \beta < 1$). If we initialize value iteration to the zero value function, the estimates of the values of these two choices are: $-\beta(1-\beta^t)/(1-\beta)$ and $-\beta^2/(1-\beta)$ at iteration $t > 1$. Therefore, value iteration will continue to choose the suboptimal action until iteration t^* where

$$\frac{-\beta(1-\beta^{t^*})}{1-\beta} < \frac{-\beta^2}{1-\beta},$$

or

$$t^* \geq \frac{\log(1-\beta)}{\log\beta} \geq \frac{1}{2}\log\left(\frac{1}{1-\beta}\right)\frac{1}{(1-\beta)} \quad .$$

Thus, in the worst case, value iteration has a run time that grows faster than $1/(1-\beta)$. □

2.4.3 Policy Iteration

Since there are $|\mathcal{A}|^{|\mathcal{S}|}$ distinct policies, and each iteration of policy iteration strictly improves the approximation [126], it is obvious that policy iteration terminates in at most an exponential number of steps.

Each step of policy iteration consists of a value-iteration-like policy-improvement step, which can be performed in $O(|\mathcal{A}||\mathcal{S}|^2)$ arithmetic operations; and a policy-evaluation step, which can be performed in $O(|\mathcal{S}|^3)$ operations by solving a system of linear

[1] Note that these rewards can be specified by $B \approx \log(\beta^2/(1-\beta)) = O(\log(1/(1-\beta)))$ bits.

equations[2]. The total run time, therefore, hinges on the number of iterations needed to find an optimal policy.

While direct complexity analyses of policy iteration have been scarce, several researchers have examined a simplified family of variations of policy iteration, which I discuss in the following section.

Sequential Improvement Policy Iteration

Whereas standard policy iteration defines the policy on step t to be

$$\pi_t(s) = \operatorname*{argmax}_a Q_t(s,a)$$

(see Table 2.3), *sequential improvement policy iteration* defines $\pi_t(s) = \pi_{t-1}(s)$ for all but one state. To be more precise, from the set of states s for which $Q_t(s, \pi_{t-1}(s)) < \max_a Q_t(s,a)$ (the previous action choice for s is no longer maximal), one is selected and $\pi_t(s)$ is set to $\operatorname{argmax}_a Q_t(s,a)$.

A detailed analogy can be constructed between the choice of state to update in sequential improvement policy iteration and the choice of pivot rule to use in simplex. Denardo [44] shows that the feasible bases for the linear program in Table 2.4 are in one-to-one correspondence with the stationary deterministic policies.

As with simplex, examples have been constructed that make sequential improvement policy iteration require an exponential number of iterations. Melekopoglou and Condon [107] examine the problem of solving cost-to-go MDPs using several variations on the sequential-improvement-policy-iteration algorithm. In a version they call *simple policy iteration*, every state is labeled with a unique index and, at each iteration, the policy is updated at the state with the smallest index of those at which the policy can be improved. They show that the family of examples suggested by Figure 2.3, from a particular starting policy, takes an exponential number of iterations to solve using simple policy iteration.

An example can be constructed for each whole number n ($n = 5$ in Figure 2.3). There are $2n$ states and they are divided into three classes: decision states (labeled s_0 through s_{n-1}), random states (labeled s'_1 through s'_{n-1}), and an absorbing state, s_n. From each decision state s_i, there are two actions available: action a_1 results in a transition to decision state s_{i+1} and action a_2 results in a transition to random state

[2] In theory, policy evaluation can be performed faster, because it primarily requires inverting a $|\mathcal{S}| \times |\mathcal{S}|$ matrix, which can be done in $O(|\mathcal{S}|^{2.376})$ time [39].

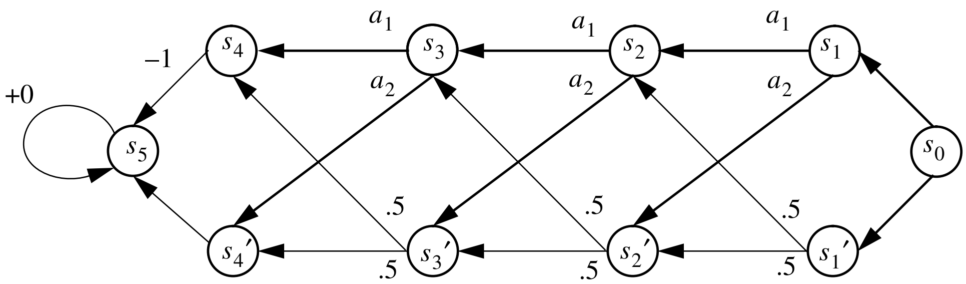

Figure 2.3: Simple policy iteration requires an exponential number of iterations to generate an optimal solution to the family of MDPs illustrated here.

s'_{i+1}. From random state s'_i, there is no choice of action and instead a random transition takes place with probability 1/2 of reaching random state s'_{i+1} and probability 1/2 of reaching decision state s_{i+1}. Actions from decision state s_{n-1} and random state s'_{n-1} both result in a transition to the absorbing state s_n. This transition has a reward of -1 in the case of decision state s_{n-1}, whereas all other transitions have zero cost.

The initial policy is $\pi_0(s) = a_1$ for all $s \in \mathcal{S}$, so every decision state s_i selects the action that takes it to decision state s_{i+1}. In the unique optimal policy, $\pi^*(s) = a_1$ for $s \neq s_{n-2}$ and $\pi^*(s_{n-2}) = a_2$. Although these two policies are highly similar, Melekopoglou and Condon show that simple policy iteration steps through 2^{n-2} policies before arriving at the optimal policy.

Although this example was constructed to hold for an undiscounted all-policies-proper criterion, it also holds under the discounted criterion regardless of discount rate. This is because the introduction of discount factor $\beta > 0$ causes the values of states s_i and s'_i to be reduced by precisely a factor of β^{n-i-1}, regardless of the policy, because every path from s_i or s'_i that includes the non-zero reward has precisely the same length. Thus, when the algorithm needs to decide whether the action for decision state s_i should be changed, the introduction of β scales down the values of the succeeding states equally, but does not change the relative order of the two choices.

Parallel Improvement Policy Iteration

When the policy is improved at all states in parallel, as in standard policy iteration, the algorithm no longer has a direct simplex analogue. It is an open question whether this can lead to exponential run time in the worst case or whether the resulting algorithm is guaranteed to converge in polynomial time. However, this version is strictly more efficient than the simple policy iteration algorithm mentioned above.

Let π_t be the policy found after t iterations of policy iteration. Let V^{π_t} be the value function associated with π_t. Let V_t be the value function found by value iteration (Table 2.2) starting with V^{π_0} as an initial value function. Puterman [126] (Theorem 6.4.6, discussed in Section B.1) showed that $V^{\pi_t}(s) \geq V_t(s)$, for all $s \in \mathcal{S}$ and therefore that policy iteration converges no more slowly than value iteration for discounted infinite-horizon MDPs. When combined with a result by Tseng [162], presented here as Lemma 2.1, that bounds the time needed for value iteration to find an optimal policy, this shows that policy iteration takes polynomial time, for a fixed discount factor. Furthermore, if the discount factor is included as part of the input as a rational number with the denominator written in unary, policy iteration takes polynomial time. This makes policy iteration a pseudopolynomial-time algorithm.

Thus, whereas policy iteration runs in polynomial time for a fixed discount factor, simple policy iteration can take exponential time, *regardless* of discount factor. This observation [96] stands in contrast to a comment by Denardo [44]. Denardo argues that block pivoting in simplex achieves the same goal as parallel policy improvement in policy iteration, and therefore that one should prefer commercial implementations of simplex to casual implementations of policy iteration. This argument is based on the misconception that one step of policy improvement on n states is equivalent in power to n iterations of simple policy iteration. In fact, one policy improvement step on n states can correspond to as many as 2^n iterations of simple policy iteration. Thus, policy iteration has not yet been ruled out as the preferred solution method for MDPs—more empirical study is needed.

2.4.4 Summary

The results of this section can be summarized by the following theorem.

Theorem 2.2 *To solve families of* MDPs *with a fixed discount factor, value iteration, policy iteration, and linear programming take polynomial time, while simple policy iteration can take exponential time. When the discount factor is included as part of the input, value iteration takes pseudopolynomial time, policy iteration takes no more than pseudopolynomial time, and linear programming takes polynomial time. No strongly polynomial-time algorithm is known.*

Proof: The theorem follows from the preceding discussion. □

2.5 Complexity Results

At this time, there is no algorithm that solves general MDPs in a number of arithmetic operations polynomial in $|\mathcal{S}|$ and $|\mathcal{A}|$ (i.e., no known algorithm is strongly polynomial). Using linear programming, however, the problem can be solved in a number of operations polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, and B, where B measures the number of bits needed to write down the transitions, rewards, and discount factor.

Papadimitriou and Tsitsiklis [116] analyzed the computational complexity of MDPs. They showed that, under discounted, average-reward, and polynomial-horizon criteria, the problem is P-complete. This means that, although it is solvable in polynomial time, if an efficient parallel algorithm were available, then all problems in P would be solvable efficiently in parallel—an outcome considered unlikely by researchers in the field. Since the linear-programming problem is also P-complete, this result implies that in terms of parallelizability, MDPs and linear programs are equivalent: a fast parallel algorithm for one would yield a fast parallel algorithm for the other. It is not known whether the two problems are equivalent with respect to strong polynomiality: although it is clear that a strongly polynomial algorithm for solving linear programs would yield one for MDPs, the converse is still open.

Papadimitriou and Tsitsiklis also show that for deterministic MDPs, optimal value functions can be found efficiently in parallel using a parallel implementation of the algorithm in Table 2.6 (i.e., the problem is in NC). This algorithm is strongly polynomial, suggesting that the presence of stochastic transitions makes MDPs harder to solve, in general.

2.6 Reinforcement Learning in MDPs

In the previous sections, I explained how to find the optimal policy for an MDP given a complete description of its states, actions, rewards, and transitions. Now I describe two reinforcement-learning algorithms for finding optimal policies from experience. Although many other reinforcement-learning algorithms have been invented, I present these two algorithms because they are easily extended to work for decision-making models in later chapters.

2.6.1 Q-learning

Q-learning [173] can be viewed as a sampled, asynchronous method for estimating the optimal state-action values, or Q function, for an unknown MDP. The most basic version of Q-learning keeps a table of values, $Q[s, a]$, with an entry for each state/action pair. The entry $Q[s, a]$ is an estimate for the corresponding component of the optimal Q function, defined by

$$Q^*(s, a) = R(s, a) + \beta \sum_{s'} T(s, a, s') V^*(s'),$$

where V^* is the optimal value function. The agent uses the experience it receives to improve its estimate, blending new information into its prior experience according to a *learning rate* $0 < \alpha < 1$.

The Q function is an ideal data structure for reinforcement learning. Recall that there are three fundamental functions in the value-iteration algorithm: the value function V, the Q function Q, and the policy π. Given a model in the form of transition and reward matrices, any of these functions can be computed from any one of the others. Without access to T and R, however, only the Q function can be used to reconstruct the other two: $V(s) = \max_a Q(s, a)$ and $\pi(s) = \operatorname{argmax}_a Q(s, a)$. In addition, the Q function is not difficult to estimate from experience.

The experience available to a reinforcement-learning agent in a Markov decision process environment can be summarized by a sequence of experience tuples $\langle s, a, r, s' \rangle$. An experience tuple is a snapshot of a single transition: the agent starts in state s, takes action a, receives reward r and ends up in state s'.

Given an experience tuple $\langle s, a, r, s' \rangle$, the Q-learning rule is

$$Q[s, a] := (1 - \alpha) Q[s, a] + \alpha \left(r + \beta \max_{a'} Q[s', a'] \right).$$

This creates a new estimate of $Q^*(s, a)$ by adding the immediate reward to the current discounted estimate of $V(s')$. Because of the way r and s' are chosen, the average value of this new estimate is exactly $R(s, a) + \beta \sum_{s'} T(s, a, s') V(s')$. In value iteration, we would assign this value directly to $Q[s, a]$. However, to get an accurate estimate, we need to average together many independent samples. The learning rate α blends our present estimate with our previous estimates to produce a best guess at $Q(s, a)$; it needs to be decreased slowly for the Q values to converge to Q^* [174, 163, 69].

In Section 3.6.3, I prove that a generalization of Q-learning converges to the optimal Q function, under certain idealized assumptions.

2.6.2 Model-based Reinforcement Learning

Using Q-learning, it is possible to learn an optimal policy without knowing T or R beforehand, and without even learning these functions en route. Although it guaranteed to find optimal policies eventually and uses very little computation time per experience, Q-learning makes extremely inefficient use of the data it gathers; it often requires a great deal of experience to achieve good performance. In model-based reinforcement learning, a model of the environment is unknown in advance, but is learned from experience. The learned model can then be used to find a good policy. This approach is especially important in applications in which computation is cheap and real-world experience costly.

For MDPs in which the state and action spaces are small enough, the learned model can be represented by three arrays, a count $C[s,a]$ of the number of times action a has been chosen in state s, a count $T_c[s,a,s']$ of the number of times this has resulted in a transition to state s', and a sum $R_s[s,a]$ of the resulting reward. Given experience tuple $\langle s,a,r,s'\rangle$, the arrays are updated by

$$\begin{aligned} T_c[s,a,s'] &:= T_c[s,a,s'] + 1 \\ R_s[s,a] &:= R_s[s,a] + r \\ C[s,a] &:= C[s,a] + 1. \end{aligned}$$

Given these statistics, we estimate

$$\tilde{T}(s,a,s') = \frac{T_c[s,a,s']}{C[s,a]} \text{ and } \tilde{R}(s,a) = \frac{R_s[s,a]}{C[s,a]}.$$

The estimated model can be used in any of several ways to find a good policy. In the *certainty-equivalence* approach [86], an optimal policy for the estimated model is found at each step. This makes maximal use of the available data at the cost of high computational overhead. In the DYNA [155], prioritized-sweeping [111] and Queue-dyna [119] approaches, an estimated value function is maintained and updated according to

$$V[s] := \max_a \left(\tilde{R}(s,a) + \beta \sum_{s'} \tilde{T}(s,a,s') V[s'] \right).$$

Because the agent has access to a model of the environment, updates can be performed at any state at any time.

The convergence of model-based reinforcement learning for MDPs was shown by Gullapalli and Barto [59]. In Section 3.6.4, I present a related theorem for a broader class of models.

Reinforcement learning is an exciting area and new algorithms and studies are appearing every day; this section barely scratches the surface of some of the more basic concepts. For more information, see the survey by Kaelbling, Littman, and Moore [74].

2.7 Open Problems

Markov decision processes have been studied intensely for almost 40 years. Even so, there are several important questions that remain unanswered.

- Is there a "clean" polynomial-time algorithm for solving MDPs, that is, one that is not dependent on general linear programming? This question was raised by Papadimitriou and Tsitsiklis [116] several years ago, and is still open.

- What is the worst-case time complexity of policy iteration? We have shown that it runs in polynomial time for a fixed discount factor, but is there a polynomial upper bound on its run time for general MDPs? Is there an exponential lower bound? What is its complexity for deterministic MDPs?

- Which of the standard algorithms is most efficient for solving the MDPs encountered in practice? Authorities appear to disagree as to whether policy iteration [126] or value iteration [44] is most effective. Recent empirical comparisons [133] appear to favor modified policy iteration. In the likely event that the best choice of algorithm depends on the structure of the MDP being solved, are there useful guidelines for choosing the best solution method given the problem?

- Is there a computable optimal or near-optimal strategy for exploring an unknown MDP? There is an elegant theory of optimal exploration via allocation indices for single-state MDPs [56]; is there some way of extending this theory to general MDPs? Or is it possible to show that the problem is somehow inherently intractable?

- There are representations for rewards and transitions that make it possible to specify compact models for MDPs with exponential-size state spaces [87, 21, 24, 113]. What are the complexity issues? It is probably computationally intractable to find ϵ-optimal policies using compact representations, but are there useful subclasses of MDPs that can be solved efficiently? This question is explored by Boutilier, Dean, and Hanks [23].

- The dual linear-programming formulation of MDPs has a flow-like interpretation. Algorithms for finding min-cost flows have been studied intensively over the last few years. Are there any flow-like algorithms that can be tailored to solve MDPs efficiently?

These questions are interesting in their own right, but take on even more importance in the context of the more advanced models addressed in later chapters. Resolution of some of these open questions would shed light on questions that arise in the more complex frameworks as well.

2.8 Related Work

In this chapter, I gave an overview of the fields of Markov decision processes and reinforcement learning. The MDP literature is quite substantial, having been gathering material for nearly 40 years. The early work of Bellman [13] and Howard [68] put the field on a firm footing. Later work by Bertsekas [15], Denardo [44], Derman [46], Puterman [126], and others, synthesized the existing material and added new results. Puterman's and Bertsekas' books [126, 16] give in-depth summaries of related work in this field.

Fundamental work in reinforcement learning and its relation to Markov decision processes and dynamic programming includes Sutton's thesis [153], which introduced temporal difference (TD) methods, of which Q-learning is a special case; Watkins' thesis [173], which developed Q-learning; and important surveys and syntheses by Barto [10], Singh [145], and others. A survey by Kaelbling, Littman, and Moore [74] gives a sense of the scope of the field.

Results on the convergence of reinforcement-learning methods in MDPs, most particularly Q-learning, are primarily due to Watkins and Dayan [174], Tsitsiklis [163], and Jaakkola, Jordan, and Singh [69]. The latter two papers brought out the connection between Q-learning and work in the field of stochastic approximation. John [71] gave a critique of the use of the asymptotic optimal policy as a target for learning.

The convergence of model-based reinforcement-learning methods was studied by Gullapalli and Barto [59]. Hernández-Lerma and Marcus [64] examined the closely related problem of finding an optimal policy for an MDP with a model specified by unknown parameters; they showed how to build an asymptotically optimal non-stationary policy for such models.

The linear programming formulation of MDPs was identified by D'Epenoux [45] and others (see Hoffman and Karp's paper [67] for a list). Kushner and Kleinman [88] explored reasons for preferring the dual formulation for some applications. Denardo [44] explicitly linked policy iteration to linear programming. Schrijver [140] provides an excellent description of the theory and complexity of linear programming.

The section on the complexity of algorithms for solving MDPs is based on the work of Condon [37], Melekopoglou and Condon [107], and Tseng [162]. Condon's interest was in a variation of the alternating Markov game model described in Chapter 4, but she recognized that many of the bounds applied to analogous MDP algorithms as well.

Papadimitriou and Tsitsiklis [116] initiated exploration into the computational complexity of solving MDPs. Their work identified deterministic models as being easier to solve, from a worst-case complexity point of view, than their stochastic counterparts.

2.9 Contributions

In this chapter I presented many of the important results concerning MDPs. I provided a new analysis of policy iteration based on earlier work by Puterman [126] and Tseng [162] which resulted in a new upper-bound analysis and new insight into the connection between policy improvement and value iteration. I extended Melekopoglou and Condon's [107] exponential lower bound for simple policy iteration to discounted MDPs, and used this result in a summary of complexity results. I derived a fast sequential algorithm for deterministic MDPs using the closed-semiring framework.

Overall, there has been extensive work on Markov decision processes by researchers in operations research, dynamic programming, complexity theory, and algorithm analysis. Existing theory and algorithms have been applied quite successfully to real-world problems in a wide array of domains (see Chapter 1 of Puterman's book [126] for a summary). In the next chapter, I will show that many of the basic results concerning MDPs hold for a much broader class of models. It is an open question whether the real-world success of MDPs will be replicated for any of the other models in this class.

Chapter 3

Generalized Markov Decision Processes

Portions of this chapter have appeared in earlier papers: "A generalized reinforcement-learning model: Convergence and applications" [97] with Szepesvári, and "Generalized Markov decision processes: Dynamic-programming and reinforcement-learning algorithms" [158] with Szepesvári.

The Markov decision process model, discussed in the previous chapter, has a number of important properties that make it easy to work with computationally. In this chapter, I introduce a new class of models that includes MDPs as a special case, and show that many of the properties of MDPs are shared by this more general class. The results are presented in an abstract framework that make them easy to generalize to other models.

3.1 Introduction

This chapter builds on the ideas from the previous chapter, generalizing them wherever possible. The model I develop here, the *generalized Markov decision process* includes all the models discussed in the thesis: MDPs, alternating Markov games, Markov games, and information-state MDPs, as well as several less-studied models. The generalized MDP model applies to several different optimality objectives: finite-horizon, all-policies-proper, expected discounted reward, and risk-sensitive discounted reward, to name a few.

The main result is that all these models have a notion of an optimal value function and an optimal policy, and that a general form of the value-iteration algorithm converges to the optimal value function. I define a notion of a greedy policy with respect to a value function for generalized MDPs and show how the Bellman error magnitude of the value function can be used to bound the suboptimality of this policy. For finite-state-space models with deterministic greedy policies, I show that value iteration identifies an optimal policy in a pseudopolynomial number of iterations.

I define a version of policy iteration for generalized MDPs, and show that the algorithm converges to an optimal policy; however, it is only useful for a subclass of generalized MDPs that obey additional properties. For a different subclass of generalized MDPs, I show that a form of Q-learning converges to the optimal Q function, under the appropriate conditions. I also show that a simple model-based reinforcement-learning algorithm converges to an optimal value function for all finite-state generalized MDPs.

3.2 Generalized Markov Decision Processes

A *generalized Markov decision process* is a tuple $\langle \mathcal{X}, \mathcal{U}, T, R, N, \bigotimes, \bigoplus, \beta \rangle$, where the fundamental quantities are a set of states $\mathcal{X}$ (perhaps infinite), a finite set of actions $\mathcal{U}$, a reward function $R : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, a transition function $T : \mathcal{X} \times \mathcal{U} \to \Pi(\mathcal{X})$, a next-state function N mapping $\mathcal{X} \times \mathcal{U}$ to finite subsets of $\mathcal{X}$, a discount factor β, a summary operator $\bigoplus$ that defines the value of transitions based on the value of the successor states, and a summary operator $\bigotimes$ that defines the value of a state based on the values of all state-action pairs.

The defining set of quantities for a generalized MDP is analogous to the defining set of quantities for an MDP, with a few noteworthy exceptions. First, the state space $\mathcal{X}$ can be infinite. To prevent this from making the transition function unwieldy, every state-action pair (x, u) can only lead to a finite set of next states $N(x, u)$. The transition probabilities for taking action u in state x are zero for all states except those in the set $N(x, u)$.

The most dramatic difference between generalized MDPs and MDPs is the appearance of the summary operators, $\bigotimes$ and $\bigoplus$. I will define them precisely; however, they can be motivated by defining the appropriate operators for the MDP model.

The equations defining the optimal value function in a MDP are, for each state s,

$$V^*(s) = \max_a \left(R(s,a) + \beta \sum_{s'} T(s,a,s') V^*(s') \right).$$

This corresponds to the fact that an optimal policy in an MDP *maximizes* the *expected* reward. In generalized MDPs, the operator $\bigotimes$ takes the place of the max over actions and the operator $\bigoplus$ takes the place of the expectation over next states. If we define

$$\bigoplus_{s'}^{(s,a)} g(s') = \sum_{s'} T(s,a,s') g(s'),$$

and we define

$$\bigotimes_a^{(s)} f(s,a) = \max_a f(s,a),$$

then we can rewrite the equations for the optimal value function in an MDP as

$$V^*(s) = \bigotimes_a^{(s)} \left(R(s,a) + \beta \bigoplus_{s'}^{(s,a)} V^*(s') \right). \tag{3.1}$$

The essence of the generalized Markov decision process framework is that whenever the operators $\bigoplus$ and $\bigotimes$ satisfy certain non-expansion properties, then Equation 3.1 is a characterization of the unique optimal value function. Both $\bigoplus$ and $\bigotimes$ are *summary operators*: $\bigoplus$ summarizes the value of a finite set of next states for each state-action pair, and $\bigotimes$ summarizes the value of a finite set of actions for each state. Not all possible definitions of $\bigoplus$ and $\bigotimes$ can be used to define a generalized MDPs; they must both be non-expansions for all states and state-action pairs, respectively. In the generalized model, the value of a state is defined by the sum of rewards along a trajectory; however, we allow other types of summaries to take the place of maximization and expectation in the MDP model.

For a summary operator $\bigodot$ to be a *non-expansion*, it must satisfy two constraints. Given functions h and h' over a finite set I,

$$\min_{i \in I} h(i) \le \bigodot_{i \in I} h(i) \le \max_{i \in I} h(i) \tag{3.2}$$

and

$$\left| \bigodot_{i \in I} h(i) - \bigodot_{i \in I} h'(i) \right| \le \max_{i \in I} |h(i) - h'(i)| \tag{3.3}$$

The first condition states that the summary of a function must lie between the largest and smallest value of the function. The second condition states that the difference

model/example reference	$\bigotimes_u^{(x)} f(x,u)$	$\bigoplus_{x'}^{(x,u)} g(x')$
disc. exp. MDP [174]	$\max_u f(x,u)$	$\sum_{x'} T(x,u,x')g(x')$
cost-based MDP [29]	$\min_u f(x,u)$	$\sum_{x'} T(x,u,x')g(x')$
evaluating policy π [154]	$\sum_u \pi(x,u)f(x,u)$	$\sum_{u'} T(x,u,x')g(x')$
alt. Markov game [26]	$\max_u$ or $\min_u f(x,u)$	$\sum_{x'} T(x,u,x')g(x')$
risk-sensitive MDP [62]	$\max_u f(x,u)$	$\min_{x' \in N(x,u)} g(x')$
evaluating risk-sensitive π	$\sum_u \pi(x,u)f(x,u)$	$\min_{x' \in N(x,u)} g(x')$
exploration-sens. MDP [71]	$\max_{\pi \in P_0} \sum_u \pi(x,u)f(\cdot)$	$\sum_{x'} T(x,u,x')g(x')$
Markov games [90]	see text	see text
information-state MDP [117]	$\max_u f(x,u)$	$\sum_{x' \in N(x,u)} T(x,u,x')g(x')$

Table 3.1: Examples of generalized Markov decision processes and their summary operators.

between the summaries of two different functions must be no larger than the largest difference between the functions.

Section C.1 explores different summary operators and proves that a broad class of operators are non-expansions. For this chapter, the most important summary operators are expectation, max, min, and the minimax operator used in Markov games. Other interesting examples include operators for computing the median, midpoint, and mean of a set of values.

It follows from the definitions above that finite-state MDPs, the continuous state-space MDPs resulting from POMDPs, and Markov games all satisfy the conditions of being a generalized MDP. Whatever we prove about generalized MDPs will apply to all of these models. In addition, several other models are in this class, some of which are not typically thought of as being related to MDPs. Table 3.1 lists some sample generalized MDPs and their summary operators. In contrast to the previous chapter, some generalized MDPs have optimal policies that are stochastic. Throughout this chapter, the notation $\pi(x,u)$ is used to signify the probability that stochastic policy π chooses action u from state x.

3.2.1 Acting Optimally

In this section I develop a notion of optimal value functions, Q functions, and policies for generalized MDPs. I casually use terms like "value function", "contraction mapping," and "fixed-point theorem" without formally developing these concepts. Discussion of the theory behind contraction mappings and the space of value functions is given in Puterman's book [126] (Appendix C and Section 6.2.2). This background is probably not necessary for an understanding of the results presented here; however, it is critical for a formal treatment of this topic.

To begin, define the dynamic-programming operator H for a generalized MDP, a function that takes a value function and creates a new value function, as

$$[HV](x) = \bigotimes_u^{(x)} \left(R(x,u) + \beta \bigoplus_{x'}^{(x,u)} V(x') \right). \tag{3.4}$$

The idea here is that $\bigoplus_{x'}{}^{(x,u)} V(x')$ is the value of the state resulting from taking action u from state x, $R(x,u) + \beta \bigoplus_{x'}{}^{(x,u)} V(x')$ is the value obtained by taking action u from state x, and $[HV](x)$ is the value of state x as obtained by one-step lookahead. For MDPs, H is the mathematical instantiation of a single step of value iteration. We can also define a dynamic-programming operator K that acts on Q functions,

$$[KQ](x,u) = R(x,u) + \beta \bigoplus_{x'}^{(x,u)} \bigotimes_{u'}^{(x')} Q(x',u').$$

To simplify the presentation of some of the later results, we sometimes treat $\bigotimes$ and $\bigoplus$ as mapping operators. To be precise, for Q function Q and value function V,

$$[\bigotimes Q](x) = \bigotimes_{u \in \mathcal{U}}^{(x)} Q(x,u),$$

and

$$[\bigoplus V](x,u) = \bigoplus_{x' \in N(x,u)}^{(x,u)} V(x').$$

Using this notation, and the obvious extension of the definitions of addition and scalar multiplication, we can express $[HV] = \bigotimes(R + \beta \bigoplus V)$ and $[KQ] = R + \beta(\bigoplus \bigotimes Q)$.

In what follows, it is helpful to establish a notion of *distance* between two value functions. For value functions V_1 and V_2, we define

$$\|V_1 - V_2\| = \sup_x |V_1(x) - V_2(x)|,$$

where $\|\cdot\|$ is a distance function known as the L_∞ norm or max norm. Intuitively, to find the distance between two value functions, we find the state where they differ the most and call the difference between values at that state the distance between the value functions. We can extend the notion of distance to cover Q functions as well. If Q_1 and Q_2 are Q functions,

$$\|Q_1 - Q_2\| = \sup_x \max_u |Q_1(x,u) - Q_2(x,u)|.$$

The non-expansion properties of $\otimes$ and $\oplus$ lead to a convenient property of these operators with regard to distances.

Lemma 3.1 *Let Q_1 and Q_2 be Q functions and V_1 and V_2 be value functions. Then $\| \bigotimes Q_1 - \bigotimes Q_2\| \leq \|Q_1 - Q_2\|$ and $\| \bigoplus V_1 - \bigoplus V_2\| \leq \|V_1 - V_2\|$.*

Proof: Using the fact that $\otimes$ and $\oplus$ are non-expansions, and Condition 3.3 for non-expansions, we have

$$\begin{aligned} \| \bigotimes Q_1 - \bigotimes Q_2\| &= \sup_x |[\bigotimes Q_1](x) - [\bigotimes Q_2](x)| \\ &\leq \sup_x \max_u |Q_1(x,u) - Q_2(x,u)| = \|Q_1 - Q_2\|, \end{aligned}$$

and

$$\begin{aligned} \| \bigoplus V_1 - \bigoplus V_2\| &= \sup_x \max_u |[\bigoplus V_1](x,u) - [\bigoplus V_2](x,u)| \\ &\leq \sup_x \max_u \max_{x' \in N(x,u)} |V_1(x') - V_2(x')| \\ &\leq \sup_{x'} |V_1(x') - V_2(x')| = \|V_1 - V_2\|. \end{aligned}$$

□

It is this distance-based bound that will be most convenient for proving results about the dynamic-programming operators H and K. Here is the first.

Lemma 3.2 *The H and K operators are contraction mappings if $\beta < 1$. In particular, if V_1 and V_2 are value functions and Q_1 and Q_2 are Q functions, $\|HV_1 - HV_2\| \leq \beta\|V_1 - V_2\|$, and $\|KQ_1 - KQ_2\| \leq \beta\|Q_1 - Q_2\|$.*

Proof: We address the H operator first. By Lemma 3.1 and the definition of H, we have

$$\begin{aligned} \|HV_1 - HV_2\| &= \| \bigotimes (R + \beta \bigoplus V_1) - \bigotimes (R + \beta \bigoplus V_2)\| \\ &\leq \|(R + \beta \bigoplus V_1) - (R + \beta \bigoplus V_2)\| \\ &\leq \beta \| \bigoplus V_1 - \bigoplus V_2\| \\ &\leq \beta\|V_1 - V_2\|. \end{aligned}$$

Lemma 3.1 and the definition of K give us

$$\begin{aligned} \|KQ_1 - KQ_2\| &= \beta \| \bigotimes [\bigoplus Q_1] - \bigotimes [\bigoplus Q_2]\| \\ &\leq \beta \| \bigoplus Q_1 - \bigoplus Q_2\| \\ &\leq \beta\|Q_1 - Q_2\|. \end{aligned}$$

□

Because the operator H is guaranteed to bring two value functions closer together, and the operator K is guaranteed to bring two Q functions closer together, they are called *contraction mappings*.

A *weighted max norm* is defined by $\|V_1 - V_2\|_w = \sup_x |V_1(x) - V_2(x)|/w(x)$ for value functions and $\|Q_1 - Q_2\|_w = \sup_x \max_u |Q_1(x,u) - Q_2(x,u)|/w(x)$ for Q functions. The introduction of the weighting function w makes it possible for states to contribute differently to the max norm; states with larger weights count less than states with smaller weights. Operator H is a contraction mapping with respect to some weighted max norm w if and only if $\|HV_1 - HV_2\|_w \leq \beta_w \|V_1 - V_2\|_w$ for some $\beta_w < 1$. In MDPs, even if the discount factor is 1, if all policies are guaranteed to reach a zero-cost absorbing state (the all-policies-proper case), then the dynamic-programming operator H is a contraction mapping with respect to some weighted max norm [18, 162]. Section C.2 provides a new proof of this fact in the context of finite-state generalized MDPs.

Lemma 3.3 *For any generalized* MDP *in which*

$$\bigoplus_{x'}^{(x,u)} g(x') = \sum_{x'} T(x,u,x') g(x'),$$

if $\beta = 1$ but all policies are guaranteed to reach a zero-reward absorbing state (the all-policies-proper case), then the H and K operators are contraction mappings with respect to some weighted max norm.

Proof: The theorem follows from the preceding discussion. □

Because all the results in this chapter are stated in terms of norms, they apply to any update rule as long as the dynamic-programming operator under consideration is a contraction mapping; in particular, they cover generalized MDPs with either $\beta < 1$ or the all-policies-proper condition. In the latter case, I abuse notation and use β to signify the constant of contraction in the appropriate weighted max norm (i.e., β_w). The fact that the optimal value functions are well defined does not imply that they are meaningful; that is, it may be the case that the optimal value function is not the same as the value function for some appropriately defined optimal policy. The results in this section apply to value functions *defined* by Bellman equations; to relate the Bellman equations to a notion of optimality, it is necessary to put forth arguments such as are given in Puterman's book [126].

Theorem 3.1 *For any generalized Markov decision process, if $\beta < 1$ then there is a unique V^*, called the* optimal value function, *such that $V^* = HV^*$; a unique Q^*, called the* optimal Q function, *such that $Q^* = KQ^*$; and an optimal (possibly stochastic) policy, π^*, such that $V^*(x) = \sum_u \pi^*(x,u)Q^*(x,u)$. This is also true if $\beta = 1$, the all-policies-proper condition holds, and an expected value criterion is used.*

Proof: Combining Lemmas 3.2 and 3.3, the H and K operators for the generalized MDP are contraction mappings with respect to some weighted max norm. The existence and uniqueness of V^* and Q^* follow directly from the Banach fixed-point theorem.

We can define the optimal value function and the optimal Q function in terms of each other:

$$V^* = \bigotimes Q^*, \tag{3.5}$$

and $Q^* = R + \beta \bigoplus V^*$. These equations can be shown to be valid from the definitions of K and H and the uniqueness of Q^* and V^*.

By Condition 3.2 of $\bigotimes$ and Equation 3.5,

$$\min_u Q^*(x,u) \leq V^*(x) \leq \max_u Q^*(x,u).$$

Therefore, it is possible to define a stochastic policy π^* such that

$$V^*(x) = \sum_u \pi^*(x,u)Q^*(x,u).$$

□

The use of the word *optimal* is somewhat strange since V^* need not be the largest or smallest value function in any sense; it is simply the fixed point of the dynamic-programming operator H. This terminology comes from the Markov decision process model, where V^* is the largest value function of all policies and is retained for consistency.

3.2.2 Exploration-sensitive MDPs

One interesting use of generalized MDPs is as a way to formalize John's [71] exploration-sensitive learning algorithm. John considered the implications of insisting that agents simultaneously act to maximize their reward *and* explore their environment; he found that better performance can be achieved if a policy incorporates the condition of persistent exploration. In John's formulation, the agent is forced to adopt a policy from a restricted set; in one example, the agent must choose a stochastic stationary policy that selects actions at random 5% of the time. The random actions ensure that the agent will experience all actions in all states infinitely often; therefore, the agent will be able to detect if its model of the environment is wrong, or if the environment changes, or the effects of any non-Markovian dependencies that may exist.

John's approach requires that the definition of optimality be changed to reflect the restriction on policies. The optimal value function is given by $V^*(x) = \sup_{\pi \in \mathcal{P}} V^\pi(x)$, where $\mathcal{P}$ is the set of permitted stationary policies, and the associated Bellman equations are

$$V^*(x) = \sup_{\pi \in \mathcal{P}} \sum_u \pi(x,u) \left(R(x,u) + \beta \sum_{x'} T(x,u,x') V^*(x') \right),$$

which corresponds to a generalized MDP model with ${\bigoplus_{x'}}^{(x,u)} g(x') = \sum_{x'} P(x,u,x') g(x')$ and ${\bigotimes_u}^{(x)} f(x,u) = \sup_{\pi \in \mathcal{P}} \sum_u \pi(x,u) f(x,u)$. Because $\pi(x,\cdot)$ is a probability distribution for any given state x, $\bigotimes$ is a non-expansion and, thus, the model can properly be considered a generalized MDP.[1]

3.3 Algorithms for Solving Generalized MDPs

The results of the previous section show that any generalized MDP has an optimal value function, Q function, and policy, and that these quantities can be defined in terms of each other. In this section, I discuss methods for finding these quantities.

3.3.1 Value Iteration

The method of *value iteration*, or successive approximations [13, 143], is a way of iteratively computing arbitrarily good approximations to the optimal value function V^*.

A single step of the process starts with an estimate, V_{t-1}, of the optimal value function, and produces a better estimate $V_t = HV_{t-1}$. I will show that applying H repeatedly causes the value function to become as close as desired to optimal.

Lemma 3.4 *Let V_t be the value function produced in the tth iteration of value iteration. After t steps of value iteration on a generalized* MDP, $\|V_t - V^*\| \leq \beta^t \|V_0 - V^*\|$.

[1] One additional condition is that the set $\mathcal{P}$ be compositional; if a separate policy is selected for each action, the combined policy must still be in $\mathcal{P}$.

Proof: We proceed by induction. The base case, $\|V_0 - V^*\| \leq \beta^0\|V_0 - V^*\|$, is self evident. By the inductive hypothesis we see

$$\|V_t - V^*\| = \|HV_{t-1} - HV^*\| \leq \beta\|V_{t-1} - V^*\| \leq \beta\beta^{t-1}\|V_0 - V^*\| = \beta^t\|V_0 - V^*\|.$$

□

In some circumstances, it is helpful to state this result without reference to the details of the initial value function V_0. Let $M = \sup_x \max_u |R(x,u)| = \|R\|$. If the agent received a reward of M on every step, its total expected reward would be $\sum_{i=0}^{\infty} \beta^i M = M/(1-\beta)$. The same result holds for the all-policies-proper case, although the reasoning is a bit different [162]. Thus, the zero value function, $V_0 = 0$ cannot differ from the optimal value function by more than $M/(1-\beta)$ at any state. This also implies that the value function for any policy cannot differ from the optimal value function by more than $2M/(1-\beta)$ at any state. This allows us to restate Lemma 3.4 in a form that bounds the number of iterations needed to find a ϵ-optimal value function.

Theorem 3.2 *Let V_0 be any value function such that $|V_0(x)| \leq M/(1-\beta)$ for all $x \in \mathcal{X}$, and let*

$$t^* = \left\lceil \frac{\log(M) + \log(\frac{1}{\epsilon}) + \log(\frac{1}{1-\beta})}{\log(\frac{1}{\beta})} \right\rceil.$$

Running value iteration for t^* or more steps results in a value function V such that $\|V - V^*\| \leq \epsilon$.

Proof: This follows from simple algebraic manipulation of the bounds given in this section. □

3.3.2 Computing Near-optimal Policies

In this section, we relate arbitrary generalized MDPs to the specific generalized MDP resulting from using the summary operator $\bigotimes_u^{\pi,(x)} f(x,u) = \sum_u \pi(x,u) f(x,u)$, where π is some stationary probabilistic policy.

Condition 3.2 can be interpreted as saying that every time the $\bigotimes$ operator is applied, it must be equivalent to applying $\bigotimes^\pi$ for some probabilistic policy π. Let V be some value function. We define the *myopic policy* with respect to V to be any $\pi : \mathcal{X} \rightarrow \Pi(\mathcal{U})$ such that

$$[HV](x) = \sum_u \pi(x,u) \left(R(x,u) + \beta[\bigoplus V](x,u) \right).$$

The existence of such a π follows from the definition of H given in Equation 3.4 and Condition 3.2; it need not be unique. Myopic policies are simply greedy policies, generalized to models in which reward is not maximized.

For a policy π, define H^π to be the dynamic-programming operator resulting from the generalized MDP that shares its state space, action space, transition function, reward function, next state function, $\bigoplus$ operator, and discount factor with the generalized MDP in question, but uses $\bigotimes^\pi$ in place of $\bigotimes$.

Because $\bigotimes^\pi$ is a non-expansion, the new model is itself a generalized MDP. Therefore, we can define V^π to be the unique value function satisfying $V^\pi = H^\pi V^\pi$, which we call the *value function for policy* π.

For MDPs, the distance between the value function for any myopic policy with respect to V and the optimal value function can be bounded as a function of the *Bellman error magnitude*, defined as $||V - HV||$; the result can be shown to apply to generalized MDPs, after a few basic results are established.

Lemma 3.5 *Let V be a value function, V^π be the value function for the myopic policy with respect to V, and V^* be the optimal value function. Let ϵ be the Bellman error magnitude for V, $\epsilon = ||V - HV||$. Then, $||V - V^\pi|| \leq \epsilon/(1-\beta)$ and $||V - V^*|| \leq \epsilon/(1-\beta)$.*

Proof: This result follows easily from the contraction property of H and the *triangle inequality*, which states that the distance from a to c can not be larger than the distance from a to b to c (for any b).

First, note that $||V - V^\pi|| \leq ||V - HV|| + ||HV - V^\pi|| = ||V - HV|| + ||H^\pi V - H^\pi V^\pi|| \leq \epsilon + \beta||V - V^\pi||$. Grouping like terms gives $||V - V^\pi|| \leq \epsilon/(1-\beta)$.

Similarly, $||V - V^*|| \leq ||V - HV|| + ||HV - V^*|| = ||V - HV|| + ||HV - HV^*|| \leq \epsilon + \beta||V - V^*||$. Grouping like terms gives $||V - V^*|| \leq \epsilon/(1-\beta)$. □

By the definitions of H and π, $HV = \bigotimes(R + \beta \bigoplus V) = \bigotimes^\pi(R + \beta \bigoplus V)$ and $H^\pi V^\pi = \bigotimes^\pi(R + \beta \bigoplus V^\pi)$. We can use these equations to help bound the distance between V^π and the V^* in terms of ϵ, the Bellman error magnitude.

Theorem 3.3 *Let V be a value function, V^π be the value function for the myopic policy with respect to V, and V^* be the optimal value function. Let ϵ be the Bellman error magnitude for V, $\epsilon = ||V - HV||$. Then, $||HV - V^\pi|| \leq \epsilon\beta/(1-\beta)$, $||HV - V^*|| \leq \epsilon\beta/(1-\beta)$, and $||V^\pi - V^*|| \leq 2\epsilon\beta/(1-\beta)$.*

Proof: The third statement follows from an application of the triangle inequality to the first two statements, which we prove now. First,

$$\|HV - V^\pi\| \quad = \quad \|H^\pi V - H^\pi V^\pi\| \leq \beta\|V - V^\pi\| \leq \epsilon\beta/(1-\beta).$$

Similarly,

$$\|HV - V^*\| \quad = \quad \|HV - HV^*\| \leq \beta\|V - V^*\| \leq \epsilon\beta/(1-\beta),$$

completing the proof. □

This result is concerned with *values* and not immediate rewards, so the total reward earned by a myopic policy is not too far from optimal. The significance of the result is that a value-iteration algorithm that stops when the Bellman error magnitude is less than or equal to $\epsilon \geq 0$ will produce a good policy with respect to ϵ.

3.3.3 Policy Iteration

In this section, I define a generalized version of policy iteration. Applied to MDPs, it is equivalent to Howard's policy-iteration algorithm [68] and applied to Markov games, it is equivalent to Hoffman and Karp's policy-iteration algorithm [67].

Unlike value iteration, the convergence of policy iteration seems to require that value is maximized with respect to some set of possible actions. To capture this, we will restrict our attention to generalized MDPs in which $\bigotimes$ can be written

$$[\bigotimes Q](x) = \max_{\rho \in \mathcal{R}} [\bigotimes^\rho Q](x) \tag{3.6}$$

where $\mathcal{R}$ is a compact set and $\bigotimes^\rho$ is a non-expansion operator for all $\rho \in \mathcal{R}$. The idea is that $\mathcal{R}$ is some set of parameters or choices from which the best choice is selected. Note that any operator can be written this way by defining $\bigotimes^\rho = \bigotimes$; the choice of parameterization ultimately determines the efficiency of the resulting policy-iteration algorithm. A generalized MDP satisfying Equation 3.6 and the monotonicity property discussed in Section 3.4.2 is called a *maximizing* generalized MDP.

The term *ρ-myopic policy* refers to a mapping $\omega : \mathcal{X} \to \mathcal{R}$ such that

$$[\bigotimes^{\omega(x)} Q](x) = \max_{\rho \in \mathcal{R}} [\bigotimes^\rho Q](x),$$

for all $x \in \mathcal{X}$. The value function for a ρ-myopic policy ω, V^ω, is defined as the optimal value function for the generalized MDP where $\bigotimes^{\omega(x)}$ is used as the summary operator in state x.

We characterize policy iteration as follows. Start with a value function V and compute its ρ-myopic policy ω and ω's value function V^ω. If $\|V - V^\omega\| \leq \epsilon$, terminate with V as an approximation of the optimal value function. Otherwise, start over, after assigning $V := V^\omega$.

We can apply this algorithm to MDPs by taking $\mathcal{R}$ to be the set of actions and $\bigotimes^\rho$ to select the Q value for the action corresponding to ρ:

$$[\bigotimes Q](x) = \max_{\rho \in \mathcal{R}} [\bigotimes^\rho Q](x) = \max_{u \in \mathcal{U}} Q(x, u).$$

Because computing V^ω is equivalent to evaluating a fixed policy and can be solved by Gaussian elimination, the resulting policy-iteration algorithm (which is just standard policy iteration) is useful. In Markov games, we take $\mathcal{R}$ to be the set of probability distributions over agent actions and $\bigotimes^\rho$ to be a minimum over opponent actions of the ρ-weighted expected Q value. As we will see in Chapter 5, computing V^ω is equivalent to solving an MDP, which is easier than finding V^* directly.

In Section 3.4.2, I argue that generalized policy iteration converges to an optimal policy under particular restrictions on $\bigotimes^\rho$.

3.4 Algorithmic Analysis

This section provides additional details on how value iteration performs on specific classes of generalized MDPs, and on the convergence of policy iteration.

3.4.1 Value Iteration

Although value iteration converges to the optimal value function (Lemma 3.4), and the suboptimality of the myopic policies generated along the way can be bounded (Theorem 3.3), it is not guaranteed to identify the optimal policy in a finite number of iterations.

Let V_t be the value function produced on the tth iteration of value iteration. This section shows that, for a certain class of finite-state-space generalized MDPs, the greedy policy with respect to V_{t^*} is optimal, for a finite t^*.

Let $\mathcal{X}$ be finite, and let B be a problem-specific parameter, for example, a bound on the number of bits needed to represent components of T and R. We say that a quantity is *polynomially bounded* if there is some polynomial in $|\mathcal{X}|$, $|\mathcal{U}|$, B, and $\log(1/(1-\beta))$ that grows asymptotically faster than that quantity. We say a quantity is pseudopolynomially bounded if it is polynomially bounded with respect to $|\mathcal{X}|$, $|\mathcal{U}|$, B, and $1/(1-\beta)$

We know that $\bigotimes$ and $\bigoplus$ have the property that for all Q and V, $[\bigotimes Q](x) = \sum_u \pi(x,u)Q(x,u)$ and $[\bigoplus V](x,u) = \sum_{x' \in N(x,u)} \tau(x,u,x')V(x')$ for some probability distributions π and τ. If, for all Q and V, π and τ can be expressed using only rational numbers with a polynomially bounded number of bits, then we say that $\bigotimes$ and $\bigoplus$ are *polynomially expressible*.

There are a few important polynomially expressible operators: maximum, minimum, selection according to a deterministic policy, and expectation according to a set of polynomially bounded probabilities. The operators in the risk-sensitive MDP model described by Heger [62] are polynomially expressible. The next theorem tells us that value iteration can be used to find optimal policies in pseudopolynomial time for models using these operators.

Theorem 3.4 *If $\mathcal{X}$ is finite, the number of bits needed to express $R(x,u)$ for all x and u is polynomially bounded, and $\bigotimes$ and $\bigoplus$ are polynomially expressible, then any myopic policy with respect to V_{t^*} is optimal, for some pseudopolynomially bounded t^*.*

Proof: Since $\bigotimes$ and $\bigoplus$ are polynomially expressible, there must be some optimal policy π^* and a function τ^* such that $[\bigotimes Q^*](x) = \sum_u \pi^*(x,u)Q^*(x,u)$ and $[\bigoplus V^*](x,u) = \sum_{x' \in N(x,u)} \tau^*(x,u,x')V^*(x')$, where the number of bits needed to express π^* and τ^* are polynomially bounded.

By an argument similar to that in the proof of Theorem 2.1, this implies that there is some polynomially bounded number B^* such that the number of bits needed to express each component of V^* and Q^* is bounded by B^*.

The rest of the argument parallels that in Lemma 2.1, using the contraction-mapping property of H and Theorem 3.3. The basic idea is that, eventually, V_t is so close to V^*, that the amount of reward lost by following a myopic policy with respect to V^t is smaller than the precision needed to specify V^*. At that point, any myopic policy must be optimal. $\square$

3.4.2 Policy Iteration

In this section, I argue that the policy-iteration algorithm defined in Section 3.3.3 converges to the optimal value function. To do this, we will need to place an additional monotonicity condition on the associated summary operators.

Consider a maximizing generalized MDP in which the optimal value function is defined by

$$V^*(x) = \bigotimes_u^{(x)} \left(R(x,u) + \beta \bigoplus_{x'}^{(x,u)} V^*(x') \right)$$

and

$$\bigotimes_u^{(x)} f(x,u) = \max_{\rho \in \mathcal{R}} \bigotimes_u^{\rho,(x)} f(x,u)$$

for some compact set $\mathcal{R}$. Assume $\bigoplus$ and $\bigotimes^\rho$ are non-expansions for all $\rho \in \mathcal{R}$. Theorem 3.1 gives conditions under which V^* is well defined. Further assume that $\bigoplus$ and $\bigotimes^\rho$ obey an additional monotonicity condition: if $g(x') \geq g'(x')$, then

$$\bigoplus_{x'}^{(x,u)} g(x') \geq \bigoplus_{x'}^{(x,u)} g'(x')$$

and similarly for $\bigotimes^\rho$.

Not all non-expansion operators satisfy the monotonicity condition. However, all the summary operators discussed in Section C.1, and therefore all the operators of immediate interest, do satisfy this additional condition. The monotonicity of these operators is proven in Section C.3.

The policy-iteration algorithm can be stated as follows. Let ω_0 be an arbitrary function mapping $\mathcal{X}$ to $\mathcal{R}$. At iteration t, let ω_{t-1} be the ρ-myopic policy with respect to V^{ω_t}. Terminate when $\|V^{\omega_t} - V^{\omega_{t-1}}\|$ is small enough.

To show that policy iteration converges, I appeal to two important results. The first is that

$$V^*(x) = \max_{\omega:\mathcal{X}\rightarrow\mathcal{R}} V^\omega(x),$$

meaning that the optimal value function dominates or equals the value functions for all possible values of ω. The second is a generalization of a result of Puterman [126] that shows that the iterates of policy iteration are bounded below by the iterates of value iteration. From these two facts, we can conclude that policy iteration converges to the optimal value function, and furthermore, that its convergence is at least as fast as the convergence of value iteration.

Theorem 3.5 *Let*

$$V^*(x) = \max_{\rho \in \mathcal{R}} \bigotimes_u^{\rho,(x)} \left(R(x,u) + \beta \bigoplus_{x'}^{(x,u)} V^*(x') \right)$$

and, for all $\omega : \mathcal{X} \rightarrow \mathcal{R}$,

$$V^{\omega}(x) = \bigotimes_{u}^{\omega(x),(x,u)} \left(R(x,u) + \beta \bigoplus_{x'}^{(x,u)} V^{\omega}(x') \right)$$

where $\bigotimes^{\rho}$ *and* $\bigoplus$ *are non-expansions and monotonic. Then, for all* $x \in \mathcal{X}$,

$$V^*(x) = \max_{\omega:\mathcal{X}\rightarrow\mathcal{R}} V^{\omega}(x).$$

Proof: This result is proven in Section C.4. □

Lemma 3.6 *Let* U_t *be the iterates of value iteration and* V_t *be the iterates of policy iteration, starting from the same initial value function. For all* t *and* $x \in \mathcal{X}$, $U_t(x) \leq V_t(x) \leq V^*(x)$.

Proof: The proof is Section C.4. □

Theorem 3.6 *If* $\mathcal{X}$ *is finite, the number of bits needed to express* $R(x,u)$ *for all* x *and* u *is polynomially bounded, and* $\bigotimes$ *and* $\bigoplus$ *are polynomially expressible, then policy iteration converges in a pseudopolynomial number of steps.*

Proof: The theorem follows from Lemma 3.6, which shows that policy iteration converges no more slowly than value iteration, combined with Theorem 3.4, which shows that value iteration converges in a pseudopolynomial number of iterations under the conditions of the theorem. □

Stronger results are available on the convergence rate of policy iteration [125, 126] that would lead to better complexity bounds than those of Theorem 3.6; however, these theorems do not appear to yield useful bounds for MDPs with finite state and action spaces. This is discussed in Section B.2.

It is worth noting that the implementation of policy evaluation in finite-state-space generalized MDPs depends on the definition of $\bigoplus$. When the expected-reward objective is used, as it is in MDPs, policy evaluation can be implemented using a linear-equation solver. When $\bigoplus$ is maximization or minimization, as it is in some games or under a risk-sensitive criterion, policy evaluation is equivalent to solving an MDP and can be accomplished using linear programming (or policy iteration!).

3.5 Complexity Results

The difficulty of solving particular generalized MDPs depends critically on the definitions of $\bigotimes$ and $\bigoplus$ and whether $\mathcal{X}$ is finite. I present complexity results for specific models in other chapters.

3.6 Reinforcement Learning in Generalized MDPs

In this section, I assume that the summary operators $\bigotimes$ and $\bigoplus$ are defined in terms of the transition function T and the reward function R. If both T and R are known in advance, the techniques I described earlier in this chapter can be used to compute optimal or near-optimal policies. In this section, I describe how reinforcement-learning algorithms can use experience to converge to optimal policies when T and R are not known in advance.

Section 2.6 described two different families of reinforcement-learning algorithms: model-free (Q-learning), and model-based. My plan in this section is to introduce a mathematical framework that captures algorithms from both of these classes, to describe a new stochastic-approximation theorem that provides conditions under which these algorithms converge, then to show how the general theorem can be applied to prove the convergence of model-free and model-based reinforcement-learning algorithms for generalized MDPs.

A mathematically more general presentation of these results is available [97, 158]; the goal in this section is to present the results as intuitively as possible, yet in sufficient generality and rigor to be useful in proving theorems presented in later chapters.

3.6.1 A Generalized Reinforcement-Learning Method

A wide variety of learning algorithms can be viewed in the following way. The algorithm begins with an initial approximation V_0 of the optimal value function V^*. With each new experience, an update rule is applied to the current approximation V_t to produce a new approximation V_{t+1}. The update rule can change as a function of time or as a function of the experience gathered.

The simplest update rule to analyze is the dynamic-programming operator H, which I discussed earlier in this chapter. To use H to compute V^*, we define $V_{t+1} = HV_t$; the resulting algorithm is value iteration, and its convergence properties were discussed in

Section 3.3.1.

As an important step towards analyzing Q-learning-like update rules, we will consider a learning rule that, in the limit, converges to HV for a fixed value function V. We consider the family of finite-state generalized MDPs with no choice of action and $\bigoplus_{x'}{}^{(x,u)} g(x') = \sum_{x'} T(x,u,x')g(x')$, that is, models with an expected value criterion. For this model, $[HV](x) = R(x,u) + \beta \sum_{x'} T(x,u,x')V(x')$, where u is the only possible action. Q-learning, applied to this model, begins with an initial value function U_0 and, given experience tuple $\langle x_t, u_t, x'_t, r_t \rangle$ at time t, defines

$$U_{t+1}(x_t) = (1 - \alpha_t(x_t))U_t(x_t) + \alpha_t(x_t)\left(r_t + \beta V(x'_t)\right),$$

and $U_{t+1}(x) = U_t(x)$ for all $x \neq x_t$. The idea behind the learning rule is that $U_t(x)$ contains an estimate of the value $R(x,u) + \beta \sum_{x'} T(x,u,x')V(x')$. If the learning rate α is decayed properly, U_t converges to HV.

We can capture the learning rule in the form of an operator

$$H_t(U,V)(x) = \begin{cases} (1 - \alpha_t(x_t))U(x) + \alpha_t(x_t)(r_t + \beta V(x'_t)), & \text{if } x = x_t \\ U(x), & \text{otherwise.} \end{cases}$$

and define

$$U_{t+1} = H_t(U_t, V). \tag{3.7}$$

Conditions for the convergence of U_t to HV are provided by classic stochastic-approximation theory [130].

A more advanced reinforcement-learning problem is computing $V^* = HV^*$, the fixed point of H, instead of the value of HV for a fixed value function. Consider the natural learning algorithm that begins with a value function V_0 and defines

$$V_{t+1} = H_t(V_t, V_t), \tag{3.8}$$

where H_t is as defined above. In a sense, this algorithm is computing HV, where V is a moving target. It combines the simple learning algorithm in Equation 3.7 with value iteration, and, as we will see shortly, converges to V^*.

In what follows, we will use $H_t(U,V)$ to stand for a generic learning rule; it is not difficult to express model-based reinforcement-learning algorithms and Q-learning-like algorithms in this form. We will also see that there are very reasonable conditions under which the learning rule captured by Equation 3.8 is guaranteed to converge to V^*.

3.6.2 A Stochastic-Approximation Theorem

The fundamental property that we will require of a sequence of operators $H_t(U, V)$ is that it can be used to approximate the value of HV by holding V fixed and iterating on U; this was illustrated in Equation 3.7. We say that H_t approximates H at V if iteration on U converges to HV with probability 1 uniformly over $\mathcal{X}$.

The following theorem shows that, under the proper conditions, we can use an operator H_t to estimate the optimal value function V^*; it is due to Szepesvári and Littman [158].

Theorem 3.7 *Let H be a contraction mapping with respect to a weighted max norm with fixed point V^*, and let H_t approximate H at V^*. Let V_0 be an arbitrary value function, and define $V_{t+1} = H_t(V_t, V_t)$. If there exist functions $0 \leq F_t(x) \leq 1$ and $0 \leq G_t(x) \leq 1$ satisfying the conditions below with probability one, then V_t converges uniformly to V^* with probability 1:*

1. *for all value functions U_1 and U_2 and all $x \in \mathcal{X}$,*

$$|(H_t(U_1, V^*))(x) - (H_t(U_2, V^*))(x)| \leq G_t(x)\|U_1 - U_2\|;$$

2. *for all value functions U and V, and all $x \in \mathcal{X}$,*

$$|(H_t(U, V^*))(x) - (H_t(U, V))(x)| \leq F_t(x)\|V^* - V\|;$$

3. *for all $k > 0$, $\Pi_{t=k}^{n} G_t(x)$ converges to zero uniformly in x as n increases; and,*
4. *there exists $0 \leq \beta < 1$ such that for all $x \in \mathcal{X}$ and large enough t,*

$$F_t(x) \leq \beta(1 - G_t(x)).$$

Proof: The theorem is proven in Section C.5. $\square$

Next, I describe some of the intuition behind the statement of the theorem and its conditions.

The iterative approximation of V^* is performed by computing $V_{t+1} = H_t(V_t, V_t)$. Because of Conditions 1 and 2, $G_t(x)$ is the extent to which the estimated value function depends on its present value and $F_t(x) \approx 1 - G_t(x)$ is the extent to which the estimated value function is based on "new" information.

In some applications, such as Q-learning, the contribution of new information needs to decay over time to ensure that the process converges. In this case, $G_t(x)$ needs to converge to one. Condition 3 allows $G_t(x)$ to converge to 1 as long as the convergence is slow enough to incorporate sufficient information for the process to converge to the right value.

Condition 4 links the values of $G_t(x)$ and $F_t(x)$ through some quantity $\beta < 1$. If it were somehow possible to update the values synchronously over the entire state space, the process would converge to V^* even when $\beta = 1$. In the more interesting asynchronous case, when $\beta = 1$, the long-term behavior of V_t is not immediately clear; it may even be that V_t converges to something other than V^*. The requirement that $\beta < 1$ ensures that the use of outdated information in the asynchronous updates does not cause a problem in convergence.

One of the most noteworthy aspects of this theorem is that it shows how to reduce the problem of approximating V^* to the problem of approximating H at V^*; in many cases, the latter is much easier to achieve and also to prove. For example, the theorem makes the convergence of Q-learning a consequence of the simpler Robbins-Monro theorem [130].

3.6.3 Generalized Q-learning for Expected Value Models

A Q-learning algorithm can be defined for the family of finite-state generalized MDPs with ${\bigoplus_{x'}}^{(x,u)} g(x') = \sum_{x'} T(x,u,x') g(x')$, that is, models with an expected value criterion. Given experience tuple $\langle x_t, u_t, x'_t, r_t \rangle$ at time t and an estimate $Q_t(x,u)$ of the optimal Q function, let

$$Q_{t+1}(x_t, u_t) := (1 - \alpha_t(x_t, u_t)) Q_t(x_t, u_t) + \alpha_t(x_t, u_t) \left(r_t + \beta \bigotimes_u^{(x'_t)} Q_t(x'_t, u) \right).$$

When ${\bigotimes_u}^{(x)} f(x,u) = \max_u f(x,u)$, this is precisely the Q-learning algorithm described in Section 2.6.1; however, a different definition of $\otimes$ captures the minimax-Q learning algorithm described in Section 5.6.1.

In this section, I derive the assumptions necessary for this learning algorithm to satisfy the conditions of Theorem 3.7 and therefore converge to the optimal Q values. The dynamic-programming operator defining the optimal Q function is

$$[KQ](x,u) = R(x,u) + \beta \sum_{x'} T(x,u,x') \bigotimes_{u'}^{(x')} Q_t(x',u')).$$

The Q-learning rule is equivalent to the approximate dynamic-programming operator

$$H_t(U,V)(x,u)$$
$$= \begin{cases} (1-\alpha_t(x_t,u_t))U(x,u) + \alpha_t(x_t,u_t)(r_t + \beta \bigotimes_u{}^{(x'_t)} V(x'_t,u)), & \text{if } x = x_t,\ u = u_t \\ U(x,u), & \text{otherwise.} \end{cases}$$

If

- $\bigotimes$ is a non-expansion and does not depend on T or R,
- x' is selected according to the probability distribution defined by $T(x,u,x')$,
- the expected value of r given x and u is $R(x,u)$,
- r has finite variance,
- every state-action pair is updated infinitely often, and
- the learning rates are decayed so that

$$\sum_{t:x_t=x,u_t=u} \alpha_t(x,u) = \infty \text{ and } \sum_{t:x_t=x,u_t=u} \alpha_t(x,u)^2 < \infty$$

  uniformly over $\mathcal{X} \times \mathcal{U}$ with probability 1,

then a standard result from the theory of stochastic approximation [130] can be used to show that H_t approximates H. That is, this method of using a decayed, exponentially weighted average correctly computes the average one-step reward.

Let

$$G_t(x,u) = \begin{cases} 1-\alpha_t(x,u), & \text{if } x = x_t \text{ and } u = u_t; \\ 0, & \text{otherwise,} \end{cases}$$

and

$$F_t(x,u) = \begin{cases} \beta\alpha_t(x,u), & \text{if } x = x_t \text{ and } u = u_t; \\ 0, & \text{otherwise.} \end{cases}$$

These functions satisfy the conditions of Theorem 3.7 (Condition 3 is implied by the restrictions placed on the sequence of learning rates α_t).

Theorem 3.7 therefore implies that the generalized Q-learning algorithm converges to the optimal Q function with probability 1. The convergence of Q-learning for discounted MDPs and alternating Markov games follows easily from this result. In addition, this result also applies to models satisfying the all-policies-proper condition by using a weighted max norm.

It is also worth noting that a Q-learning-type algorithm can be defined for generalized MDPs under a worst-case-reward criterion [62]. Theorem 3.7 can be used to prove the convergence of this algorithm [158].

3.6.4 Model-based Methods

The fundamental assumption of reinforcement learning is that the reward and transition functions are not known in advance. Although Q-learning shows that optimal value functions can sometimes be estimated without ever explicitly learning R and T, learning R and T makes more efficient use of experience at the expense of additional storage and computation. The parameters of R and T can be learned from experience by keeping statistics on the expected reward for each state-action pair and the proportion of transitions to each next state for each state-action pair. In model-based reinforcement learning, R and T are estimated on-line, and the value function is updated according to the approximate dynamic-programming operator derived from these estimates. Theorem 3.7 can be used to prove the convergence of a wide array of model-based reinforcement-learning methods.

In this section, we assume that $\bigoplus$ may depend on T and/or R, but $\bigotimes$ may not. Although this is the more common case, it is possible to extend the argument below to allow $\bigotimes$ to depend on T and R as well.

In model-based reinforcement learning, R and T are estimated by the quantities R_t and T_t, and $\bigoplus^t$ is an estimate of the $\bigoplus$ operator defined using R_t and T_t. As long as every state-action pair is visited infinitely often, there are a number of simple methods for computing R_t and T_t that converge to R and T. A bit more care is needed to ensure that $\bigoplus^t$ converges to $\bigoplus$, however. For example, in expected-reward models, ${\bigoplus_{x'}}^{(x,u)} g(x') = \sum_{x'} T(x,u,x') g(x')$ and the convergence of T_t to T guarantees the convergence of $\bigoplus^t$ to $\bigoplus$. On the other hand, in worst-case-reward models,

$\bigoplus_{x'}{}^{(x,u)} g(x') = \min_{x':T(x,u,x')>0} g(x')$ and it is necessary to approximate T in a way that ensures that the set of x' such that $T_t(x,u,x') > 0$ converges to the set of x' such that $T(x,u,x') > 0$. This can be accomplished easily, for example, by setting $T_t(x,u,x') = 0$ if no transition from x to x' under u has been observed.

Assuming T and R can be estimated in a way that results in the convergence of $\bigoplus^t$ to $\bigoplus$, the approximate dynamic-programming operator H_t defined by

$$H_t(U,V)(x) = \begin{cases} \bigotimes_u{}^{(x)} \left(R_t(x,u) + \beta \bigoplus_{x'}^{t,(x,u)} V(x') \right), & \text{if } x \in \tau_t \\ U(x), & \text{otherwise,} \end{cases}$$

converges to H with probability 1 uniformly. Here, the set $\tau_t \subseteq \mathcal{X}$ represents the set of states whose values are updated on step t; one popular choice is to set $\tau_t = \{x_t\}$. Other algorithms use a larger τ_t set to speed up learning: DYNA [156] supplements τ_t with a randomly generated set of states while prioritized sweeping [111] and Queue-DYNA [119] use heuristics to select elements for τ_t that will result in the fastest possible convergence of the value function.

The functions

$$G_t(x) = \begin{cases} 0, & \text{if } x \in \tau_t; \\ 1, & \text{otherwise,} \end{cases}$$

and

$$F_t(x) = \begin{cases} \beta, & \text{if } x \in \tau_t; \\ 0, & \text{otherwise,} \end{cases}$$

satisfy the conditions of Theorem 3.7 as long as each x is in infinitely many τ_t sets (Condition 3) and the discount factor β is less than 1 (Condition 4).

As a consequence of this argument and Theorem 3.7, model-based methods can be used to find optimal policies in MDPs, alternating Markov games, Markov games, risk-sensitive MDPs, and exploration-sensitive MDPs.

3.7 Open Problems

The exploration of this class of models has just begun. Although generalized MDPs were developed to generalize the specific models used in this thesis, they may be worthy of independent study.

- Although many relevant summary operators have been shown to have the required non-expansion properties, is there a better, more succinct, or more intuitive characterization of the summary operators used in generalized MDPs?
- Can generalized MDPs be extended to infinite action spaces?
- Some natural summary operators, like Boltzmann weighting (Section C.1), do not have the non-expansion property. Is there a way to characterize these operators and the effect of using them in value iteration? In the case of Boltzmann weighting, there are examples where the H operator has multiple fixed points.
- Model-free reinforcement-learning updates appear to require the use of the expected reward summary operator. Is there a general theory of how to take a given definition of the $\bigoplus$ operator and create an appropriate model-free reinforcement-learning algorithm?
- The class of generalized MDPs was developed primarily with regard to a discounted reward criterion. Is it possible to extend the results to the average reward criterion? Would that be interesting?
- Is it possible to extend Sutton's TD(λ) algorithm [154] to the generalized MDP model?

3.8 Related Work

This chapter's main function is to prove some basic properties of MDPs and variants of MDPs. Puterman [126] develops the results for MDPs with finite state and action spaces, but also more general spaces. His work focuses exclusively on maximizing expected reward. Van Der Wal [166] addresses a generalized set of objective criteria for MDPs and Markov games.

The inspiration for trying to find a uniform framework for these proofs grew out of the independent work of Shapley [143] on games and of Howard [68] and Bellman [13] on MDPs. Both of these efforts developed the value-iteration algorithm and proved its convergence. This chapter attempted to capture the essence of both approaches in a unified way.

There are many, many models that satisfy the conditions of being a generalized MDP; very few of these are interesting. In the remaining chapters of this thesis, I examine a few models of interest, but there are others that are worth mentioning. John [71] looks at MDPs that maximize expected reward *given* that actions are chosen with respect to a perpetually exploring policy. His learning rule for the uniform-exploration case can be shown to be a generalized MDP and, hence, inherits the results proven in this chapter.

Heger [62, 63] has developed a collection of results, including a proof of Theorem 3.3, for risk-sensitive MDPs: generalized MDPs that maximize worst-case reward. The results in this chapter extend some of Heger's results. I prove finite convergence of value iteration and policy iteration for his minimax criterion, and make it possible to extend risk-sensitive objective criteria to infinite state spaces and games.

The results on pseudopolynomial convergence of some generalized MDPs comes directly from the work of Tseng [162] for MDPs and Condon [36] for alternating Markov games. The idea of bounding the greedy policy according to an approximate value function is common knowledge in the dynamic-programming community, and was introduced to the reinforcement-learning community by Williams and Baird [180] and Singh and Yee [147].

The work presented in Section 3.6 is closely related to several previous research efforts. Szepesvári [157] described a generalized reinforcement-learning model, and used it to define a set of conditions under which there is an optimal policy that is stationary, and when it can be found as the myopic policy with respect to the optimal value function. The specific generalized MDP model presented here is both more and less general than Szepesvári's model; however, Theorem 3.7 is useful in both frameworks.

Jaakkola, Jordan, and Singh [69] and Tsitsiklis [163] developed the connection between stochastic-approximation theory and reinforcement learning, focusing on the MDP model. The mathematics and insight used in Theorem 3.7 are not substantially different from that used in the earlier papers; however, the form of Theorem 3.7 makes it particularly convenient for proving the convergence of reinforcement-learning algorithms. Concretely, Theorem 3.7 shows that, given a contraction mapping H and an idea of how to approximate HV^*, it is often fairly easy to design algorithms that approximate V^* itself.

Waldmann [172] developed a highly general model of dynamic-programming problems, with a focus on deriving approximation bounds. Vendu and Poor [167] introduced a class of abstract dynamic-programming models that is far more comprehensive than the model discussed here. In addition to permitting non-additive operators and value functions with values from any set (not just the real numbers), they showed how, in the context of finite-horizon models, a weaker "commutativity" condition can replace the monotonicity condition exploited in this chapter.

3.9 Contributions

In this chapter, I presented a new model, which I called generalized Markov decision processes, for the purpose of making it easier to present background results that are common to all the models covered in this thesis. The model, which defines optimal values by a simple generalization of the Bellman equation, might be useful to researchers studying other types of sequential decision making. I proved a series of concrete results concerning the model, including the contraction of the dynamic-programming operator, the convergence of value iteration and policy iteration, the convergence of a model-free reinforcement-learning algorithm, the convergence of a model-based reinforcement-learning algorithm. I also gave a simple new proof that all-policies-proper MDPs result in contraction with respect to some weighted max norm, and described a new stochastic-approximation theorem, developed in collaboration with Szepesvári [158].

The generalized MDP framework highlights common elements among several different sequential decision-making models, and extends existing models in an interesting way. In the following chapters, I examine several simple applications of the results from this chapter, but there are a number of interesting directions yet to be explored.

Chapter 4

Alternating Markov Games

Portions of this chapter and the next have appeared in earlier papers: "Markov games as a framework for multiagent reinforcement learning" [90], and "An introduction to reinforcement learning" [74] with Kaelbling and Moore.

Game playing has dominated the artificial-intelligence world as a problem domain ever since the field was born. Two-player games do not fit into the established MDP framework because the optimality criterion for games is typically not one of maximizing reward in the face of a fixed environment, but one of maximizing reward against an optimal adversary. Nonetheless, there are profound similarities between the problem of finding an optimal policy for an MDP and that of finding an optimal policy for a game.

4.1 Introduction

In this chapter, I review some of the important similarities and differences between MDPs and two-player games in which players alternate moves (alternating Markov games). In the next chapter, I consider a more general class of games in which both players choose their moves simultaneously (Markov games). Both chapters address only zero-sum games, that is, games in which reward for one player comes directly "out of the pocket" of the other.

Interest in finding optimal policies for games is spread over several different fields: complexity theorists have linked the (open) question of the existence of polynomial-time algorithms for finding optimal policies for alternating games to the equivalence of particular Turing-machine models [36]; reinforcement-learning researchers have adapted MDP-based learning algorithms to a very general class of games [90] and many researchers have used reinforcement learning in these environments; economists and game theorists [168, 166, 143] have studied Markov games as a model for understanding the behavior of individuals in multiagent systems.

4.2 Alternating Markov Games

In this chapter, I describe alternating Markov games, in which stochastic control of the state transitions alternates between an agent and its opponent. This includes most standard board games like backgammon, chess, and tic-tac-toe, but also captures more complex situations in which rewards are issued throughout the interaction. The identity of the player in control of the transition is part of the state description, and control does not necessarily change hands after every action.

4.2.1 Basic Framework

In its general form, a Markov game, sometimes called a stochastic game [114], is defined by a set of states, $\mathcal{S}$, and a *collection* of action sets, $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$, one for each agent in the environment. State transitions are a stochastic function of the current state and one action from each agent: $T(s, a_1, a_2, \ldots, a_k, s')$ is the probability of a transition from s to s' when agent 1 chooses $a_1 \in \mathcal{A}_1$, agent 2 chooses $a_2 \in \mathcal{A}_2$, etc. Agent i also has an associated reward function, $R_i(a_1, a_2, \ldots, a_k)$, and attempts to maximize its expected sum of discounted rewards, $E\{\sum_{j=0}^{\infty} \beta^j r^i_{t+j}\}$, where r^i_{t+j} is the reward received j steps into the future by agent i.

In this chapter and the next, I consider a well-studied specialization of Markov games in which there are only two agents and they have diametrically opposed goals. This makes it possible to represent the agents' instantaneous rewards with a single reward function that one agent seeks to maximize and the other, called the *opponent*, seeks to minimize. The set $\mathcal{A}_1$ denotes the agent's action set, and $\mathcal{A}_2$ denotes the opponent's action set. In this chapter, only one agent has an action choice in each state; $\mathcal{S}_1$ signifies the states in $\mathcal{S}$ in which the agent has a choice of action, and $\mathcal{S}_2$ signifies the other states. It is not necessary to assume that control strictly alternates between the two players; Section D.1 shows that, from a complexity standpoint, such an assumption does not change the class of models considered. The function $R(a, s)$ denotes the immediate reward to the agent for taking action $a \in \mathcal{A}_1$ in state $s \in \mathcal{S}_1$ or the immediate reward to the agent for its opponent taking action $a \in \mathcal{A}_2$ in state $s \in \mathcal{S}_2$.

Restricting the model to two-player zero-sum games simplifies the mathematics but makes it impossible to consider important phenomena such as cooperation. Nonetheless, the present model subsumes MDPs, which are just alternating Markov games in which $|\mathcal{A}_2| = 1$ or $|\mathcal{S}_2| = 0$. In the next chapter, I consider a generalization of alternating Markov games in which the players select their moves synchronously.

4.2.2 Acting Optimally

As in Chapter 2, an optimal policy is one that maximizes the expected sum of discounted reward. There are subtleties in applying this objective to Markov games, however. First, consider the parallel scenario in MDPs.

In an MDP, an optimal policy is one that maximizes the expected sum of discounted reward; it is *undominated*, meaning that there is no state from which any other policy can achieve a better expected sum of discounted reward. Every MDP has at least one optimal policy, and of the optimal policies for a given MDP, at least one is stationary and deterministic. This means that, for any MDP, there is a policy $\pi : \mathcal{S} \to \mathcal{A}$ that is optimal. The policy π is called stationary because it does not change as a function of time, and it is called deterministic because the same action is always chosen whenever the agent is in state s, for all $s \in \mathcal{S}$.

For many games, there is no policy that is undominated because performance depends critically on the choice of opponent. How, then, can we define an optimal policy? In the game-theory literature, this difficulty is resolved by evaluating each policy with respect to the opponent that makes it look the worst. This performance measure prefers conservative strategies that can force any opponent to a stalemate over more daring ones that accrue a great deal of reward against some opponents and lose a great deal to others. This is the essence of minimax: Behave so as to maximize your reward in the worst case.

Given this definition of optimality, alternating Markov games share several important properties with MDPs: every alternating Markov game has a non-empty set of optimal policies, at least one of which is stationary and deterministic [36].

As in MDPs, the discount factor, β, can be thought of as the probability that the game will be allowed to continue after the current move, i.e., $1-\beta$ is the probability that a zero-value forced draw will be proclaimed on any given move.

Another connection between alternating Markov games and MDPs is that, if we hold the opponent's policy fixed, the agent faces a stationary environment and any of the MDP algorithms of Chapter 2 can be used to find an optimal counter strategy. This fact will be helpful in deriving an efficient policy-iteration algorithm for alternating Markov games.

4.2.3 Simple Stochastic Games

Condon [36] reduced alternating Markov games to their simplest possible form, which she called "simple stochastic games." In this model, there are four kinds of states: states in which the agent deterministically controls the transitions, states in which the opponent deterministically controls the transitions, states in which neither player controls the transition but instead a transition is made to one of two states with equal probability, and absorbing "win" states (one for each player) that end the game when they are reached. The model includes a single transition with a non-zero reward, no discount factor, two actions per state, and only deterministic transitions and probability 1/2 transitions. Nonetheless, it is possible to show that any alternating Markov game with rational immediate rewards and transition probabilities can be transformed to an equivalent simple stochastic game with at most a polynomial increase in problem size [36, 183].

Although the simple stochastic game model is elegant, its connection to the traditional MDP is somewhat indirect; I will focus on the alternating Markov game model, although the results I present apply to simple stochastic games as well.

4.3 Algorithms for Solving Alternating Markov Games

In this section, I review methods for finding optimal policies for alternating Markov games. The algorithms here are all variations of algorithms for solving Markov decision processes.

4.3.1 Value Iteration

In an MDP, given $Q^*(s, a)$, an agent can maximize its reward using the "greedy" strategy of always choosing the action with the highest Q value. This strategy is greedy in the sense that it treats $Q^*(s, a)$ as a surrogate for immediate reward and then acts to maximize its immediate gain. It is optimal because the Q function is an accurate summary of future rewards.

A similar observation can be exploited in alternating Markov games. First, we redefine $V^*(s)$ to be the expected reward to the agent for following the optimal minimax policy against an optimal opponent starting from state s, and $Q^*(s, a)$ to be the expected reward for the agent taking action a (if $s \in \mathcal{S}_1$) or the opponent taking action a (otherwise) and both players continuing optimally thereafter. Then the value of a state $s \in \mathcal{S}$ in an alternating Markov game is

$$V^*(s) = \begin{cases} \max_{a_1 \in \mathcal{A}_1} Q^*(s, a_1) & \text{if } s \in \mathcal{S}_1 \\ \min_{a_2 \in \mathcal{A}_2} Q^*(s, a_2) & \text{otherwise,} \end{cases}$$

and the value of a state-action pair (s, a) is

$$Q^*(s, a) = R(s, a) + \beta \sum_{s'} T(s, a, s') V^*(s').$$

The resulting recursive equations look much like the equations for Q and V in MDPs, and indeed the analogous value-iteration algorithm converges to the correct values [36].

4.3.2 Policy Iteration

Policy iteration in Markov decision processes proceeds by alternating between computing the value of the current policy and finding the greedy policy for the current value function. In alternating Markov games, there are essentially two active policies at any given time, and as a result, there are several choices for generalizing policy iteration to alternating Markov games.

Table 4.1 gives a generic policy-iteration algorithm for alternating Markov games. It follows the MDP algorithm quite closely, alternating between policy evaluation and policy improvement, and makes use of two important subroutines: `evalGame` and `improvePoliciesGame`. The `evalGame` subroutine, given in Table 4.2, simply computes the value function that results from the agent following policy π_1 and the opponent

```
PolicyIterationGame(M = ⟨S_1, S_2, A_1, A_2, T, R, β⟩) := {
    foreach s ∈ S_1 π_1(s) := a, for some a ∈ A_1
    foreach s ∈ S_2 π_2(s) := a, for some a ∈ A_2
    V_0 := evalGame(π_1, π_2, M)
    t := 0
    loop
        t := t + 1
        (π_1, π_2) := improvePoliciesGame(π_1, π_2, V_{t-1}, M)
        V_t := evalGame(π_1, π_2, M)
    until V_{t-1}(s) = V_t(s) for all s
    return (π_1, π_2)
}
```

Table 4.1: The policy-iteration algorithm for alternating Markov games.

```
evalGame(π_1, π_2, ⟨S_1, S_2, A_1, A_2, T, R, β⟩) := {
    Solve the following system of linear equations:
        find: v[s]
        s.t.: v[s] = R(s, π_1(s)) + β Σ_{s'∈S} T(s, π_1(s), s')v[s'], for all s ∈ S_1
        and: v[s] = R(s, π_2(s)) + β Σ_{s'∈S} T(s, π_2(s), s')v[s'], for all s ∈ S_2
    return v
}
```

Table 4.2: Computing the value function for a given pair of policies.

following policy π_2; as in MDPs, the value function is computed by solving a system of linear equations.

At this high level, the policy-iteration algorithm is identical to the one described in Chapter 2. The difference is in the implementation of `improvePoliciesGame`. How should we choose new policies for the players that are closer to the optimal policies?

Let V_{t-1} be a value function and π_1 and π_2 be policies for the agent and the opponent. There are at least four sensible choices for constructing policies π_1' and π_2' that are improvements relative to V_{t-1}:

1. let π_1' and π_2' both be greedy with respect to V_{t-1};
2. let π_1' be greedy with respect to V_{t-1}, and π_2' be the optimal policy for the opponent *given* that the agent is following π_1';
3. let π_2' be greedy with respect to V_{t-1}, and π_1' be the optimal policy for the opponent *given* that the agent is following π_2';
4. let π_1' be the optimal policy for the agent given that the opponent is following π_2, and let π_2' be the optimal policy for the opponent given that the agent is following π_1.

These choices are not all equivalent; in fact, only choices 2 and 3, which are duals, lead to algorithms that converge in general [37]. We therefore base the implementation of our `improvePoliciesGame` subroutine in Table 4.3 on choice 2. Since we need π_2 to be the optimal counter-strategy to the greedy π_1, Table 4.4 shows how to compute the optimal counter-strategy for a fixed policy. The basic idea is that, once one player's actions are fixed, only one player is left with any choice of action; the resulting model is an MDP. The algorithms in Table 4.4 make use of MDP policy iteration to solve the resulting one-player game, though any of the MDP algorithms from Chapter 2 would suffice. In the routine for computing an optimal policy for the opponent given a fixed policy for the agent, the rewards are negated; this is because the opponent's job is to *minimize* reward and the MDP algorithms from the previous chapter *maximize* reward.

The value function computed in the process of finding the optimal π_2 given π_1 is the same value function that is found when evaluating the resulting policies. A more efficient implementation would save this value function instead of throwing it away and recomputing it.

improvePoliciesGame$(\pi_1, \pi_2, V, \mathcal{M} = \langle \mathcal{S}_1, \mathcal{S}_2, \mathcal{A}_1, \mathcal{A}_2, T, R, \beta \rangle) := \{$
    foreach $s \in \mathcal{S}_1$
        $\pi_1(s) := \text{argmax}_{a \in \mathcal{A}_1} (R(s,a) + \beta \sum_{s' \in \mathcal{S}} T(s,a,s')V(s'))$
    $\pi_2 :=$ counterStratGame2$(\pi_1, \mathcal{M})$
    return (π_1, π_2)
$\}$

Table 4.3: Computing improved policies for both players.

counterStratGame2$(\pi_1, \langle \mathcal{S}_1, \mathcal{S}_2, \mathcal{A}_1, \mathcal{A}_2, T, R, \beta \rangle) := \{$
    foreach $s \in \mathcal{S}$ and $s' \in \mathcal{S}$ and $a \in \mathcal{A}_2$ $\{$
        if $(s \in \mathcal{S}_2)$ $T'(s,a,s') := T(s,a,s')$
        else $T'(s,a,s') := T(s,\pi_1(s),s')$
    foreach $s \in \mathcal{S}$ and $a \in \mathcal{A}_2$ $R'(s,a) := -R(s,a)$
    return(PolicyIterationMDP$(\langle \mathcal{S}_1 \cup \mathcal{S}_2, \mathcal{A}_2, T', R', \beta \rangle)$)
$\}$

counterStratGame1$(\pi_2, \langle \mathcal{S}_1, \mathcal{S}_2, \mathcal{A}_1, \mathcal{A}_2, T, R, \beta \rangle) := \{$
    foreach $s \in \mathcal{S}$ and $s' \in \mathcal{S}$ and $a \in \mathcal{A}_1$ $\{$
        if $(s \in \mathcal{S}_1)$ $T'(s,a,s') := T(s,a,s')$
        else $T'(s,a,s') := T(s,\pi_2(s),s')$
    return(PolicyIterationMDP$(\langle \mathcal{S}_1 \cup \mathcal{S}_2, \mathcal{A}_1, T', R, \beta \rangle)$)
$\}$

Table 4.4: Computing the optimal counter-strategy for player 2 given a policy for player 1, and vice versa.

4.3.3 Polynomial-time Algorithms for Simple Games

There is no algorithm that is known to solve general alternating Markov games in polynomial time, although it is easy to believe that such an algorithm exists [37]. This section examines algorithms that provably solve simplified classes of alternating Markov games in polynomial time.

Cycle-free Games

We say that an alternating Markov game is *cycle free* if, aside from designated zero-reward absorbing states, there is absolutely no way that any state can be revisited. Games with a non-renewable resource, such as spaces on the board in tic-tac-toe or Connect-Four, are cycle free. Cycle-free games are easy to solve, because no more than $|\mathcal{S}_1| + |\mathcal{S}_2|$ steps can elapse before the absorbing state is reached. These games can be solved in polynomial time using value iteration, or by a procedure referred to as DAG-SP [28], which I will describe now.

In a cycle-free game, all states can be categorized by the largest possible number of transitions that can elapse between an agent occupying the state and the agent reaching an absorbing state. Let $d(s)$ be the maximum number of transitions (distance) from s to an absorbing state. We can define d by

$$d(s) = \begin{cases} 0 & \text{if } s \text{ is absorbing,} \\ 1 + \max_{s',a} I\{T(s,a,s') > 0\} d(s') & \text{otherwise.} \end{cases}$$

Here $I\{e\}$ is the indicator function for boolean expression e; $I\{e\} = 1$ if e is true, and 0 otherwise. We are guaranteed that $d(s) \leq |\mathcal{S}_1| + |\mathcal{S}_2|$ for all s in non-cycle games. In addition, if s' is reachable from s in a single transition, $d(s') < d(s)$. As a result, an algorithm can solve the Bellman equations by assigning values to states in order of increasing $d(s)$.

Of course, it is not necessary to compute and sort these distances explicitly; a topological sort [40] of the transition graph accomplishes the same purpose much more easily.

Deterministic Goal-reward Games

In many games, like checkers, it is possible to return to the same board configuration over and over again. The DAG-SP algorithm from the previous section can not be applied to these games. However, checkers can be characterized as a *deterministic goal-reward game* and can be solved efficiently (relative to the astronomical size of its state space!) by an algorithm closely related to Dijkstra's shortest-path algorithm [40].

Like cycle-free games, reward-goal games have a set of absorbing states. Unlike cycle-free games, the only non-zero rewards in a reward-goal game are issued immediately upon entering an absorbing state. This means that the optimal values in deterministic reward-goal games can be conveniently characterized; there is at most one non-zero reward reached in any game. The optimal value for any state in a deterministic reward-goal game is either zero or can be written $\beta^k r$, where r is one of the non-zero rewards, and k is the number of steps before the non-zero reward is reached.

We can solve deterministic goal-reward games efficiently by carefully working backwards from the absorbing states. At an intuitive level, this is accomplished by taking the largest reward and propagating it backwards to states in $\mathcal{S}_1$ (the maximizing states). Similarly, the most negative reward can be propagated backwards through the states in $\mathcal{S}_2$ (the minimizing states). Once this process gets stuck, the remaining states all have value zero.

More precisely, the algorithm begins by defining $V(s) = 0$ for all absorbing states and leaving it undefined otherwise. Define a lower bound l and upper bound u on the value of each state as follows. If $s \in \mathcal{S}_1$, $l(s)$ is the value of

$$\max_{a \in \mathcal{A}_1}(R(s,a) + \beta V(N(s,a))),$$

where the maximization is over actions such that $V(N(s,a))$ is defined; recall that N is the next-state function. If $V(N(s,a))$ is undefined for all a, $l(s)$ is undefined. Because the value $l(s)$ is attainable for some action, the true value of $V(s)$ is at least this large. For $s \in \mathcal{S}_2$, $u(s)$ is defined analogously.

If $s \in \mathcal{S}_1$, we can compute an upper bound on $V(s)$ as follows. We define an optimistic Q value to be

$$Q(s,a) = R(s,a) + \beta V(N(s,a))$$

if $V(N(s,a))$ is defined. If $V(N(s,a))$ is not defined, $Q(s,a) = \max_{s'} \max_d \beta^d V(s')$ where the maximization of s' is over all s' such that $V(s')$ is defined, and the maximization over d is over path lengths from s to s' when a is taken as the first action. If $V(s') < 0$, this quantity is maximized when $d = \infty$. The value $u(s) = \max_a Q(s,a)$ is an upper bound on the value of $V(s)$ because the Q values are computed optimistically. For $s \in \mathcal{S}_2$, $l(s)$ is defined analogously.

If, for any s, $V(s)$ is undefined and $l(s) = u(s)$, then we can define $V(s) = l(s)$. This will happen if $V(N(s,a))$ is defined for all a, or one of the actions for which $V(N(s,a))$ is defined dominates the optimistic estimate for all the other actions. This can happen, for example, when $s \in \mathcal{S}_1$ and s is one step away from the largest defined $V(s')$.

Each time a new $V(s)$ is defined, the upper and lower bounds need to be recomputed. If, at any time, there is no s for which $V(s)$ is undefined and $l(s) = u(s)$, we can set all the undefined $V(s)$ values to zero. This can be justified by induction on the value of d in the definition of the optimistic bounds above, but intuitively, each of the states for which $V(s)$ is undefined would prefer to be in a zero-reward cycle to any other outcome they could ensure.

A straightforward implementation of this algorithm in which the upper and lower bounds are recomputed from scratch at each iteration runs in polynomial time. However, a more efficient implementation can be created by storing the optimistic Q values in priority queues. This novel algorithm was inspired by Condon's [36] algorithm for deterministic simple stochastic games and Boyan and Moore's [28] application of Dijsktra's algorithm to deterministic MDPs. Condon's algorithm is a great deal simpler, but is only defined for undiscounted games with a unique goal state.

Deterministic Constant-reward-cycle Games

The algorithm of the previous section can be extended using ideas from the cycle-free-game algorithm to solve a wider class of games in polynomial time.

In a constant-reward-cycle game, for every possible cycle of states there exists a value r such that every immediate reward on that cycle is exactly r. In the goal-reward games of the previous section, r is zero. In other games, several different values of r are possible depending on the cycle. These games can be solved by clustering the states according to whether they can participate in any cycles, and, if so, the value of r for those cycles. Note that all the actions a from state s that can result in a cycle must have the same immediate reward.

The *transition graph* for a deterministic alternating Markov game is the graph consisting of one node for each state, and one directed edge for each state-action pair. There is an edge in the transition graph from node s to node s' if there is some action

a for which $N(s, a) = s'$. We can partition the states of the game by their strongly connected components [40] in the transition graph. Each component consists of either a single state or a set of states in which every pair of states in the set is involved in a cycle. Thus, by assumption, each component c has a single immediate-reward value, which we write as r_c.

We will label each of the nodes of the graph with its value. Begin by making $V(s)$ undefined for all states s. As in the cycle-free-game algorithm, consider each component c in reverse topological order, i.e., starting with the components that can reach no other components. For all states within a given component, all rewards are equal, except for those that result in a transition out of the component. Values can be assigned to all the states in the component using a variation of the deterministic goal-reward algorithm of the previous section.

It is not obvious that this algorithm has any practical application over that of the goal-reward algorithm. However, it is interesting in that it is the most general polynomial-time algorithm known for solving alternating Markov games.

4.3.4 Other Algorithms

Condon [37] surveyed algorithms for the simple stochastic game model. The results of Zwick and Paterson [183] and Condon [36] show that the discounted alternating Markov game model is polynomially reducible to the simple stochastic game model; this means that any of the simple stochastic game algorithms can also be used to solve alternating Markov games, indirectly. However, most of the algorithms in Condon's paper can be applied to alternating Markov games with little or no change.

In addition to algorithms designed for alternating games, any algorithm that can solve general Markov games can solve alternating Markov games as well. Chapter 5 discusses these more general algorithms.

4.4 Algorithmic Analysis

The correctness of and run time bounds for the algorithms in Section 4.3 follow from the analogous algorithms for generalized Markov decision processes (Chapter 3).

Lemma 4.1 *Alternating Markov games are a type of generalized Markov decision process.*

Proof: To show that alternating Markov games are a form of generalized MDP, we need to define the state space, action space, reward and transition functions, and optimality equations. Let $\mathcal{X} = \mathcal{S}_1 \cup \mathcal{S}_2$, $\mathcal{U} = \mathcal{A}_1 \times \mathcal{A}_2$, $R(x, (a_1, a_2)) = R(x, a_1)$ if $s \in \mathcal{S}_1$ and $R(x, a_2)$ otherwise, and $T(x, (a_1, a_2), x') = T(x, a_1, x')$ if $s \in \mathcal{S}_1$ and $T(x, a_2, x')$ otherwise. The optimality equations are

$$V^*(s) = \begin{cases} \max_{a_1 \in \mathcal{A}_1} Q^*(s, a_1), & \text{if } s \in \mathcal{S}_1 \\ \min_{a_2 \in \mathcal{A}_2} Q^*(s, a_2), & \text{otherwise,} \end{cases} \tag{4.1}$$

and

$$Q^*(s, a) = R(s, a) + \beta \sum_{s'} T(s, a, s') V^*(s').$$

The relevant summary operators are shown to be non-expansions in Section C.1. $\square$

4.4.1 Value Iteration

Lemma 4.1 and Theorem 3.4 together imply that the value-iteration algorithm for alternating Markov games converges to the optimal value function.

In addition, Condon [36] shows that the complexity (in terms of bits) of the optimal value function is bounded by a polynomial in the size of the description of the game. As discussed in Theorem 3.4, this fact can be used to show that value iteration can be used to find optimal policies in pseudopolynomial time, or polynomial time for fixed discount factor, for discounted alternating Markov games and all-policies-proper alternating Markov games.

4.4.2 Policy Iteration

To apply the policy-iteration algorithm for generalized MDPs to alternating Markov games, the optimality equations (Equation 4.1) must be rewritten so that the outermost operator is a maximization. This can be accomplished as follows:

$$V^*(s) = \max_{a_1 \in \mathcal{A}_1} \begin{cases} Q^*(s, a_1), & \text{if } s \in \mathcal{S}_1 \\ \min_{a_2 \in \mathcal{A}_2} Q^*(s, a_2), & \text{otherwise,} \end{cases}$$

and

$$Q^*(s, a) = R(s, a) + \beta \sum_{s'} T(s, a, s') V^*(s').$$

The resulting policy-iteration algorithm is exactly the algorithm of Section 4.3.2, in which policy evaluation is accomplished by solving a minimum-reward MDP. A dual algorithm can be obtained by placing the minimization on the outside and performing policy evaluation using a maximum-reward MDP.

By Lemma 3.6, the policy-iteration algorithm for games converges no more slowly than value iteration, finding an optimal policy and value function in a pseudopolynomial number of iterations.

4.4.3 Linear Programming

No one has yet been able to reduce the problem of solving an alternating Markov game to that of solving a polynomial-size linear program. This is somewhat surprising because, as with MDPs, the optimal value function for an alternating Markov game is the solution to a polynomial-size set of linear equations. In addition, there is a very natural linear program that would seem to solve this problem perfectly.

In this section, I describe the linear program that seems to solve alternating Markov games, and give a simple example that shows why it does not. This issue is also treated in a paper by Condon [37].

The optimal value function V for an alternating Markov game satisfies the Bellman equations in Equation 4.1. Section 2.3.3 described a linear program that solved a similar set of equations. In that formulation, the maximization operator is implemented as the least upper bound; that is, there is a constraint demanding that $V(s)$ is greater than or equal to the one-step value for each action, and the objective function minimizes $V(s)$. An analogous technique can be used to implement the minimum operator.

This leads to the natural assumption that these two techniques could be combined to create a linear program whose solution is exactly the value of an alternating Markov game. Table 4.5 provides the "algorithm" suggested by this idea.

Unfortunately, the objective function cannot be used to jointly ensure that the maximization and minimizations are implemented properly. Figure 4.1 depicts a 4-state deterministic alternating Markov game for which the linear-programming algorithm in Table 4.5 does not work.

In this example, states s_1 and s_1' are nominally controlled by the agent and state s_2 is nominally controlled by the opponent. However, the single actions available from states s_1 and s_1' lead to state s_2, and the single action available from state s_2 leads to an absorbing state. In the corresponding linear program, $V(s_1) \geq 3/4\, V(s_2)$, $V(s_1') \geq 3/4\, V(s_2)$, $V(s_2) \leq 1$, and $V(s_1) + V(s_1') - V(s_2)$ needs to be minimized. Although it

```
gameLP(⟨S1, S2, A1, A2, T, R, β⟩) := {
```

gameLP$(\langle \mathcal{S}_1, \mathcal{S}_2, \mathcal{A}_1, \mathcal{A}_2, T, R, \beta\rangle) := \{$

Solve the following linear program:

minimize: $\sum_{s\in\mathcal{S}_1} v[s] - \sum_{s\in\mathcal{S}_2} v[s]$

s.t.: $v[s] \geq R(s,a) + \beta \sum_{s'\in\mathcal{S}} T(s,a,s')v[s']$, for all $s \in \mathcal{S}_1$ and $a \in \mathcal{A}_1$

and: $v[s] \leq R(s,a) + \beta \sum_{s'\in\mathcal{S}} T(s,a,s')v[s']$, for all $s \in \mathcal{S}_2$ and $a \in \mathcal{A}_2$

variables: $v[s]$ for all $s \in \mathcal{S}$

return v

$\}$

Table 4.5: Trying to solve an alternating Markov game via linear programming. This algorithm is incorrect.

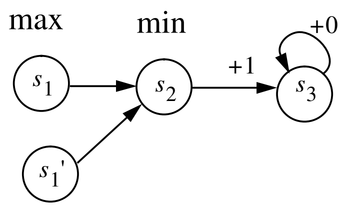

Figure 4.1: A small deterministic alternating Markov game for which the natural linear-programming formulation does not specify the optimal value function ($\beta = 3/4$).

is true that the optimal value function ($V(s_1) = 3/4$, $V(s_1') = 3/4$, $V(s_2) = 1$) satisfies the constraints and has an objective value of $1/2$, the incorrect value function $V(s) = 0$ for all s also satisfies the constraints and has a smaller objective value.

This shows that the naive application of linear programming to the problem of solving alternating Markov games (even deterministic ones) is incorrect. Although there are simple modifications that can be applied to repair this example, no general solution is known; it is possible that the problem can be formulated and solved as a linear program, but no one has yet found a way to do this.

4.5 Complexity Results

Condon [36] showed that, like MDPs, simple stochastic games can be solved by finding an optimal value function and that the optimal value function can be written as the solution to a polynomial-size set of linear equations. This is roughly because the solution to a simple stochastic game can be expressed as a pair of deterministic stationary policies (one for each player) that are in *equilibrium*, meaning that neither player has any incentive to change its policy if the other player's policy stays fixed.

From these observations, it is relatively straightforward to see that the problem of finding a minimax optimal policy is in the complexity class NP∩co-NP [36]. This is because we can guess a policy for either player and verify its optimality in polynomial time using a polynomial-time algorithm for solving the resulting MDP.

There are very few problems that are in NP∩co-NP and yet are not known to be solvable in polynomial time. Unfortunately, this is one of them. Although there are dozens of natural algorithms for solving the problem, Condon [37] showed that almost all of them are either wrong or run in exponential time in the worst case.

Condon shows that the deterministic-transition version of the simple stochastic game problem can be solved in polynomial time. Zwick and Paterson [183] tried to extend this result by considering deterministic alternating Markov games, that is, games with general rewards and discounting but only deterministic transitions. They expressed confidence that this problem is in P, but were unable to prove it. They did show that the discounted version is solvable in pseudopolynomial time using an argument closely related to the proof that value iteration is pseudopolynomial for MDPs.

Section 4.3.3 gave a polynomial time algorithm for a subclass of deterministic alternating Markov games in which every cycle (a sequence of transitions from a state back to itself) consists of identical immediate rewards. Although this broadens the class of games known to be solvable in polynomial time, there is still a great deal of room for improvement.

The polynomial-horizon version of general alternating Markov games is P-complete. P-hardness follows easily from the analogous result for MDPs, proven by Papadimitriou and Tsitsiklis [116] and value iteration can be used to solve polynomial-horizon games in polynomial time. However, unlike MDPs, the problem remains P-hard even when all transitions are deterministic. This can be shown by an easy reduction from the monotone circuit-value problem [57]—essentially, the opponent takes the place of the stochastic transitions in Papadimitriou and Tsitsiklis' MDP proof [116].

4.6 Reinforcement Learning in Alternating Games

As mentioned in the introduction, game playing is one of the best studied domains for reinforcement learning. One application, well ahead of its time, was Samuel's checkers playing system [134]; it employed a training scheme similar to the updates used in value iteration and Q-learning. Tesauro [160] used the TD(λ) algorithm [154] to find an excellent policy for backgammon.

Tesauro's work is interesting for many reasons. I include a brief description here for its motivational appeal. Backgammon has approximately 10^{20} states, making table-based reinforcement learning virtually impossible. Instead, Tesauro used a back-propagation-based three-layer neural network as a function approximator for the value function, mapping board position to an estimate of the probability of victory for the current player. Basic TD-Gammon used very little pre-defined knowledge of the game, and the representation of a board position was a direct encoding, sufficiently powerful only to permit the neural network to distinguish between conceptually different positions. The more advanced TD-Gammon was provided with the same raw state information supplemented by a number of hand-crafted features of backgammon board positions. The results have been exceptional.

Although experiments with other games have in some cases produced interesting learning behavior, no success close to that of TD-Gammon has been repeated. Other games that have been studied include Go [139] and Chess [161]. It is an open question as to how the success of TD-Gammon might be repeated in other domains.

The main challenges these projects face, however, are in designing algorithms that can deal with the huge state spaces that results from formalizing traditional board games as alternating Markov games. The alternating-Markov-game approach is most appropriate for problems with undecomposable state spaces and general reward functions.

How does one go about using reinforcement learning to solve an alternating Markov game? Because of the many similarities between alternating Markov games and MDPs, researchers have simply used variations of existing reinforcement-learning algorithms to solve games. In the next sections, I show that this is perfectly acceptable, as long as updates are performed correctly.

4.6.1 Simple Minimax-Q Learning

The Q-learning update rule for MDPs can also be applied to alternating Markov games: $Q[s,a] := (1-\alpha)Q[s,a] + \alpha(r + \beta V(s'))$ for experience tuple $\langle s, a, r, s'\rangle$. This learning rule converges to the optimal Q function assuming that every action is experienced in every state infinitely often and that new estimates are blended with previous ones using a slow enough exponentially weighted average (see Section 3.6.3). The major difference is that

$$V(s') = \begin{cases} \max_{a'_1 \in \mathcal{A}_1} Q(s', a'_1) & \text{if } s' \in \mathcal{S}_1 \\ \min_{a'_2 \in \mathcal{A}_2} Q(s', a'_2) & \text{otherwise,} \end{cases}$$

whereas in MDPs, it is a simple maximization.

The algorithm is a generalization of Q-learning, and existing convergence results do not directly apply. It is also a special case of the minimax-Q learning algorithm, described in the next chapter. The convergence theorem for generalized Q-learning, stated in Section 3.6.3, applies to simple minimax-Q learning as a consequence of Lemma 4.1.

Theorem 4.1 *Simple minimax-Q learning converges to the optimal Q values with probability 1 under the appropriate conditions.*

Proof: The theorem follows from the results in Section 3.6.3. □

4.6.2 Self-play Approach

There are many other ways of adapting MDP-oriented reinforcement-learning algorithms to Markov games [26]. Some take advantage of the fact that, often, a complete transition model for the game is known in advance, making sampled updates unnecessary; others exploit the agent-opponent symmetry that is present in many games by storing $Q[s,a]$ values for $s \in \mathcal{S}_1$ only and noting that $Q^*(s_1, a_1) = -Q^*(s_2, a_2)$ when (s_1, a_1) and (s_2, a_2) are symmetric state-action pairs (described below). Many of these approaches are special cases or simplifications of simple minimax-Q learning and their convergence to optimal minimax policies follows from Theorem 4.1.

The *self-play* algorithm can be applied to the class of symmetric, alternating Markov games. In these games the action space for the two agents is the same, $\mathcal{A}_1 = \mathcal{A}_2$, and the state space can be split into a set $\mathcal{S}_1$ of states stochastically controlled by the agent, and a set $\mathcal{S}_2$ of states stochastically controlled by the opponent. The sizes of $\mathcal{S}_1$ and $\mathcal{S}_2$ are the same and all transitions from states in $\mathcal{S}_1$ ($\mathcal{S}_2$) have zero probability of remaining in $\mathcal{S}_1$ ($\mathcal{S}_2$). Furthermore, we can define a "board flipping function" f that maps each $s_1 \in \mathcal{S}_1$ to some $s_2 \in \mathcal{S}_2$ and vice versa. The flipping function has the property that $R(s,a) = -R(f(s), a)$, and $T(s,a,s') = T(f(s), a, f(s'))$. This is just a complicated way to say that any move for the agent can be turned into into an identical move for the opponent and vice versa.

Because of the symmetry in this class of games, the optimal value function V^* and Q function Q^* satisfy $V^*(s) = -V^*(f(s))$ and $Q^*(s,a) = -Q^*(f(s), a)$, for any state s and action a. An easy way to see this is to notice that the zero value function satisfies these properties and that they are preserved by a step of value iteration. This suggests that the self-play algorithm need only maintain Q values for the states in $\mathcal{S}_1$. Using the symmetry properties, we can write the updates as:

$$Q[s_1, a] := (1-\alpha)Q[s_1, a] + \alpha\left(r - \gamma \min_{a'} Q[f(s_2), a']\right)$$

for a transition from s_1 to s_2 and

$$Q[f(s_2), a] := (1-\alpha)Q[f(s_2), a] + \alpha\left(r - \gamma \min_{a'} Q[s_1, a']\right)$$

for a transition from s_2 to s_1.

It is easy to see that each update is precisely a simple minimax-Q learning update with the values of $\mathcal{S}_1$ being updated in different ways. The fact that some states are updated "out of order" just means that they are being updated more often but still in the proper way; this can only improve convergence. In applications in which the model is known in advance [160], it is not even necessary to represent the Q values explicitly; instead, the value function can be modified directly, as in value iteration.

Theorem 4.2 *Self-play algorithms converge to the optimal Q values with probability 1 under the appropriate conditions.*

Proof: This follows fairly easily from the results of Section 3.6.3. $\square$

4.6.3 Non-converging Update Rules

There are approaches to learning games that do not converge in general. In this section, I examine an approach that treats the opponent as part of the stochastic environment.

Consider a game in which all transitions, except to an absorbing goal state, result in zero reward. Imagine that the agent is in state s_1. After taking an action a_1, the resulting state is s_2 and control belongs to the opponent. The opponent now takes an action, bringing the state to s'_1 and returns control back to the agent. Under the simple minimax-Q learning rule, two updates are performed,

$$Q[s_1, a_1] := (1-\alpha)Q[s_1, a_1] + \alpha \left(\beta \min_{a'_2} Q[s_2, a'_2]\right),$$

and

$$Q[s_2, a_2] := (1-\alpha)Q[s_2, a_2] + \alpha \left(\beta \max_{a'_1} Q[s'_1, a'_1]\right).$$

However, from the agent's point of view, there was only one transition—from state s_1 to state s'_1 via action a_1. This implies a single update,

$$Q[s_1, a_1] := (1-\alpha)Q[s_1, a_1] + \alpha \left(\beta \max_{a'_1} Q[s'_1, a'_1]\right).$$

If the opponent chooses its actions according to a fixed policy, this update rule will converge to the value of the optimal counter-policy, and not the minimax optimal policy. If the opponent adopts a non-stationary policy, the update rule will not necessarily converge to anything meaningful.

One of the main results of this section, then, is that it is possible to learn optimal minimax strategies for games using reinforcement learning. The popular *self-play* method, in which a system learns about a game by playing it against itself for a long time, can be shown to converge to an optimal strategy as long as the simple minimax-Q learning update rule is used, and the system visits all possible game configurations often enough.

4.7 Open Problems

The most glaring open problems with respect to alternating Markov games involve the existence of polynomial-time algorithms.

- Can alternating Markov games be solved in polynomial time?
- What if we restrict ourselves to deterministic alternating Markov games? We know that by making the problem any simpler, for example, by restricting rewards to be zero except upon entering an absorbing state, polynomial-time algorithms exist, so the deterministic problem is in a perfect position to be solved.
- Like alternating Markov games, the problem of deciding whether a given number is prime is in the class NP∩co-NP. Primes can be recognized by a randomized algorithm in polynomial time with a bounded probability of error. Perhaps a randomized algorithm for alternating Markov games would be easier to find. There is a randomized subexponential-time algorithm [101]; is there one that runs in polynomial time?
- A connection can be made between deterministic MDPs and min-cost flow problems (see Chapter 2). Can these connections be exploited to find an efficient algorithm for alternating Markov games?

4.8 Related Work

The study of games has been divided among several different disciplines: game theory, reinforcement learning, and computational complexity. Although there has been some cross fertilization between these fields, many of the fundamental results have been discovered separately by individual researchers in the different areas.

The study of the computational properties of games in the game-theory literature dates back at least to the work of von Neumann and Morgenstern [168], which addressed solutions to the single-state simultaneous-action games known as matrix games. Shapley [143] extended these concepts to multi-stage Markov games. Surveys of Markov games from a game-theory perspective have been written by Van Der Wal [166] and Vrieze [170]. A shorter survey is also available in a game-theory overview edited by Peters and Vrieze [120]. Filar [54] specifically examined the difference between simultaneous- and alternating-action games.

It is interesting to note that many of the great minds of computer science worked on creating game-playing programs. Russell and Norvig's artificial intelligence textbook [132] lists contributions by Babbage, Zermelo, Von Neumann, Wiener, Shannon, Turing, and Knuth.

Alternating Markov games have been the source of a great deal of attention in the reinforcement-learning world. One of the earliest systems for game playing was Samuel's checker-playing program [134], which improved with experience and was inspired by many of the same insights that underlie simple minimax-Q learning. More recent examples of learning in alternating Markov games include Tesauro's backgammon player [160]; Boyan's backgammon and tic-tac-toe players [26]; Schraudolph, Dayan and Sejnowski's Go player [139]; and Thrun's chess player [161].

In the complexity and algorithms literature, Condon [36] initiated the study of simple stochastic games, "the simplest possible restriction of Shapley's model, which retains just enough complexity so that no polynomial time algorithm is known." The model is essentially an undiscounted alternating Markov game with restricted transition probabilities and action sets and rewards only of plus and minus one upon transition to a zero-reward absorbing state. Condon showed that solving a "stopping" (all-policies-proper) simple stochastic game is actually equivalent to solving an alternating Markov game. She describes connections from this problem to Markov games, as well as to important open problems in complexity theory. Later work [37] examined algorithmic approaches to the problem, with the hope of finding a polynomial-time algorithm to solve it. Although this problem is still open, Ludwig [101] was able to show that the problem of finding the optimal value function for a simple stochastic game can be solved in subexponential time. Zwick and Paterson [183] examined deterministic average-reward and deterministic discounted games and showed that these problems are no harder than solving simple stochastic games, that pseudopolynomial-time algorithms exist, and that no polynomial-time algorithms are known.

4.9 Contributions

In this chapter, I described a generalization of Markov decision processes to a type of multiagent environment called an alternating Markov game. I proved a new theorem showing that strictly alternating Markov games are just as hard to solve as alternating Markov games. I described the extension of value iteration and policy iteration to games, and explained that no polynomial-time algorithm is known for solving this class of models. I derived a new algorithm for solving constant reward-cycle alternating Markov games in polynomial time, by combining the core algorithmic ideas of two previous polynomial-time algorithms. I showed, for the first time, that reinforcement-learning algorithms developed for alternating Markov games converge to optimal minimax policies.

It seems inevitable that a polynomial-time algorithm for alternating Markov games will be found. There are important algorithmic ideas that have been recently discovered in the context of solving min-cost flow problems and hard combinatorial optimization problems, and some of these ideas are likely to be useful in finding provably efficient algorithms for alternating Markov games. This would settle one of the more intriguing open problems in the area of sequential decision making, and perhaps spark interest in developing useful applications.

Chapter 5

Markov Games

Markov games, also called stochastic games, are a model of sequential decision making that both predates and generalizes Markov decision processes. The topic was originally studied by Shapley [143]. This chapter generalizes the previous chapter by considering two-player Markov games in which rewards and transitions are determined by the simultaneous actions of both players.

5.1 Introduction

Most board games (chess, checkers, tic-tac-toe, etc.) are played by people taking turns changing the state of the game. This form of game is very convenient for humans to play because it requires no hidden information or implied trust; all players have access to all information at all times.

There are familiar conflict situations that have a more *simultaneous* quality to them. In football, for example, the offensive and defensive coaches call plays without knowing what the other coach will do. In hockey, a player taking a penalty shot decides whether to shoot high or low, while the goalie commits to blocking one or the other type of shot. In business, MCI decides to start an ad campaign defending itself against whatever negative claims AT&T might be making. All these examples have the property that the two decision makers choose a course of action that becomes immediately apparent to the other decision maker; decisions are made in the face of information that is complete except for the current decision of the other player. This is a different kind of information structure from alternating games, in which nothing is hidden, and incomplete-information games like poker in which information remains hidden through a sequence of decisions. It is these simultaneous-action complete-information games that are the subject of this chapter.

5.2 Markov Games

The set of Markov games subsumes both Markov decision processes and alternating Markov games, described in previous chapters, as special cases.

5.2.1 Basic Framework

As before, $\mathcal{A}_1$ and $\mathcal{A}_2$ represent the action choices available to the agent and its opponent. Instead of partitioning the state space according to which player has control of transitions, here the players control the transitions together. The functions $R(s, a_1, a_2)$ and $T(s, a_1, a_2, s')$ represent the immediate rewards to the agent and transition probabilities resulting from the agent taking action $a_1 \in \mathcal{A}_1$ and the opponent taking action $a_2 \in \mathcal{A}_2$ from state s.

An important special case is when $|\mathcal{S}| = 1$. The resulting game is called a *matrix game*; it is the earliest form of game studied in the game-theory literature [168]. The name "matrix game" comes from the fact that the relevant parameters can be summarized by a $|\mathcal{A}_1| \times |\mathcal{A}_2|$ matrix consisting of the immediate reward values. Solving a matrix game is known to be polynomially equivalent to solving a linear program [47].

5.2.2 Acting Optimally

As with alternating Markov games, I mainly consider the problem of finding minimax-optimal policies. Once again, I consider only the discounted expected value criterion. It is possible to define a notion of undiscounted rewards for Markov games, but not all Markov games have optimal strategies in the undiscounted case [114]. This is because, in some games, it is best to postpone risky actions indefinitely but not to avoid them forever.

Like MDPs and alternating Markov games, every Markov game has a stationary optimal policy. Unlike the other models, however, there are Markov games with no deterministic optimal policy. A classic example is "Rock, Paper, Scissors," in which any deterministic policy can be consistently defeated, whereas the optimal stochastic policy always breaks even. The need for stochastic action choice stems from the agent's

		Agent		
		rock	paper	scissors
Opponent	rock	0	1	-1
	paper	-1	0	1
	scissors	1	-1	0

Table 5.1: The matrix game for "Rock, Paper, Scissors."

uncertainty of its opponent's current action and its requirement to avoid being "second guessed."

Like alternating Markov games, when one player's policy is held fixed, the other player's optimal counter strategy can be found as the solution to an MDP.

5.3 Algorithms for Solving Markov Games

In this section, I briefly review methods for finding optimal policies for Markov games using extensions of the value-iteration and policy-iteration algorithms. Both methods rely on a subroutine for finding the optimal stochastic policy to a matrix game.

5.3.1 Matrix Games

In this section, I describe the problem of finding an optimal policy for a single-state Markov game. Although this type of game is a very special case, the linear program used to solve it forms the basis for multi-state algorithms.

A *matrix game* is defined by the matrix R of immediate rewards. Component $R(a_1, a_2)$ is the reward to the agent for choosing action a_1 when the opponent chooses action a_2. The agent's goal is to choose actions to maximize its expected reward while the opponent's goal is to minimize it. Table 5.1 gives the matrix game corresponding to the well-known game of "Rock, Paper, Scissors."

The agent's policy ρ is a probability distribution over the actions in $\mathcal{A}_1$. For "Rock, Paper, Scissors," ρ is made up of 3 components: ρ[rock], ρ[paper], and ρ[scissors]. Under a minimax criterion, the optimal agent's minimum expected reward should be as large as possible. How can we find a policy that achieves this? Imagine that we would be satisfied with a policy that is guaranteed an expected score of v no matter which action the opponent chooses. The inequalities in Table 5.2, with $\rho \geq 0$, constrain the components of ρ to represent exactly those policies—any solution to the inequalities

$$\begin{array}{rcccccl} & & \rho[\text{paper}] & - & \rho[\text{scissors}] & \geq v & \textit{(vs. rock)} \\ - & \rho[\text{rock}] & & + & \rho[\text{scissors}] & \geq v & \textit{(vs. paper)} \\ & \rho[\text{rock}] & - \ \rho[\text{paper}] & & & \geq v & \textit{(vs. scissors)} \\ & \rho[\text{rock}] & + \ \rho[\text{paper}] & + & \rho[\text{scissors}] & = 1 & \end{array}$$

Table 5.2: Linear constraints on the solution to a matrix game.

`matrixLP`$(\mathcal{A}_1, \mathcal{A}_2, R) := \{$
    Solve the following linear program:
        maximize: v
        s.t.: $v \leq \sum_{a_1 \in \mathcal{A}_1} \rho[a_1] R(a_1, a_2)$, for all $a_2 \in \mathcal{A}_2$
        and: $\rho[a_2] \geq 0$, for all $a_2 \in \mathcal{A}_2$
        and: $\sum_{a_2 \in \mathcal{A}_2} \rho[a_2] = 1$
        variables: v, $\rho[a_2]$ for all $a_2 \in \mathcal{A}_2$
    return ρ
$\}$

Table 5.3: Solving a matrix game via linear programming.

would suffice.

For ρ to be optimal, we must identify the largest v for which there is some value of ρ that makes the constraints hold. Linear programming can be used solve this problem; in this example, it finds a value of 0 for v and (1/3, 1/3, 1/3) for ρ. We can abbreviate the general linear program as

$$v = \max_{\rho \in \Pi(\mathcal{A}_1)} \min_{a_2 \in \mathcal{A}_2} \sum_{a_1 \in \mathcal{A}_1} R(a_1, a_2)\rho[a_1],$$

where $\Pi(\mathcal{A}_1)$ represents the set of probability distributions over $\mathcal{A}_1$, and

$$\sum_{a_1} R(s, a_1, a_2)\rho[a_1]$$

expresses the expected reward to the agent for adopting stochastic policy ρ against the opponent's action a_2. Table 5.3 gives a subroutine for computing an optimal policy for the agent in a matrix game.

The value found by the subroutine in Table 5.3 is the largest reward that the agent can guarantee itself; this is sometimes called the maximin value. An alternate definition is for the value to be the smallest amount of reward the opponent can force the agent to get; this is sometimes called the minimax value. An important result about matrix games, as well as the more general Markov games, is that the maximin and minimax values are equal [168, 143]. Using the maximin definition, the agent is the only one that needs to choose actions stochastically, because once the agent's policy is fixed, the opponent faces a simple minimization problem (or MDP, in the Markov-game case) that can be optimized by a deterministic policy.

5.3.2 Value Iteration

In MDPs and alternating Markov games, the problem of finding an optimal policy can be reduced to that of finding the optimal Q values. The same is true for general Markov games, although the process of extracting the optimal policy from the optimal Q values is somewhat more complex.

Define $V^*(s)$ to be the expected reward to the agent when both players follow minimax optimal policies starting from state s. Define $Q^*(s, a_1, a_2)$ to be the expected reward to the agent taking action a_1 when the opponent chooses a_2 from state s and both players continue optimally thereafter. The optimal choice of action from state s, then, is one that maximizes $Q^*(s, a_1, a_2)$ with respect to the minimizing choice of a_2. This problem is identical to the problem of solving a matrix game, discussed in Section 5.3.1.

The value of a state $s \in \mathcal{S}$ in a Markov game is

$$V^*(s) = \max_{\rho \in \Pi(\mathcal{A}_1)} \min_{a_2 \in \mathcal{A}_2} \sum_{a_1 \in \mathcal{A}_1} Q^*(s, a_1, a_2)\rho[a_1],$$

and the Q value of action a_1 against action a_2 in state s is

$$Q^*(s, a_1, a_2) = R(s, a_1, a_2) + \beta \sum_{s'} T(s, a_1, a_2, s')V^*(s').$$

The optimal policy for the agent in state s is to choose actions according to the probability distribution $\pi(s, \cdot)$ that maximizes

$$\min_{a_2 \in \mathcal{A}_2} \sum_{a_1 \in \mathcal{A}_1} Q^*(s, a_1, a_2)\pi(s, a_1).$$

The resulting recursive equations look much like the Bellman equations for Q^* and V^* in MDPs, and indeed the analogous value-iteration algorithm converges to the correct values [114]. Unlike MDPs however, the greedy policy is not necessarily optimal after any finite number of steps.

```
PolicyIterationMarkovGame(M = ⟨S, A1, A2, T, R, β⟩, ε) := {
    foreach s ∈ S V0(s) := 0
    t := 0
    loop
        t := t + 1
        (π1, π2, Vt) := improvePoliciesMarkovGame(Vt−1, M)
    until maxs |Vt−1(s) − Vt(s)| < ε
    return π1
}
```

Table 5.4: The policy-iteration algorithm for Markov games.

```
improvePoliciesMarkovGame(V, M = ⟨S, A1, A2, T, R, β⟩) := {
    foreach s ∈ S {
        foreach a1 ∈ A1 and a2 ∈ A2
            Q(a1, a2) := R(s, a1, a2) + β Σs'∈S T(s, a1, a2, s')V(s')
        π1(s, ·) := matrixLP(A1, A2, Q)
    }
    (π2, V') := counterStratMarkovGame2(π1, M)
    return (π1, π2, V')
}
```

Table 5.5: Computing improved policies for both players.

5.3.3 Policy Iteration

The policy-iteration algorithm for alternating Markov games described in Chapter 4 extends to Markov games. As in alternating Markov games, not all possible definitions of policy improvement lead to a convergent algorithm. An important difference between alternating Markov games and Markov games is that there is no finite-size set of policies that is known to include an optimal policy. As a result, policy iteration produces a sequence of better and better policies, but will not necessarily converge in finite time.

Tables 5.4, 5.5, and 5.6 give subroutines for finding a near-optimal policy for a Markov game via policy iteration. The underlying ideas follow those developed in Section 4.3.2.

```
counterStratMarkovGame2(π₁, ⟨S, A₁, A₂, T, R, β⟩) := {
    foreach s ∈ S and s' ∈ S and a₂ ∈ A₂
        T'(s, a₂, s') := Σ_{a₁} π₁(s, a₁)T(s, a₁, a₂, s')
    foreach s ∈ S and a₂ ∈ A₂ R'(s, a₂) := − Σ_{a₁} π₁(s, a₁)R(s, a₁, a₂)
    return (PolicyIterationMDP(⟨S, A₂, T', R', β⟩))
}
```

Table 5.6: Computing the optimal counter-strategy for a fixed policy.

5.4 Algorithmic Analysis

The most important analytic tool for Markov games is expressed in the following lemma.

Lemma 5.1 *Markov games are a type of generalized* MDP.

Proof: The Bellman equations given in Section 5.3.2 look a bit different from the earlier examples of generalized MDPs. Nonetheless, as shown in Section C.1, the Markov game summary operator,

$$\bigotimes_{(a_1,a_2)} f(a_1, a_2) = \max_{\rho\in\Pi(\mathcal{A}_1)} \min_{a_2\in\mathcal{A}_2} \sum_{a_1\in\mathcal{A}_1} \rho[a_1] f(a_1, a_2)$$

is a non-expansion. □

An important result concerning the analysis of algorithms for Markov games is that the optimal value function (and policy) need not consist of rational numbers, even if the components of the transition matrix, reward matrix, and the discount factor are all rational. This result is discussed in more detail in Section 5.5, but it is important to note this now because the following analyses makes use of it.

5.4.1 Matrix Games

As I mentioned earlier, solving matrix games is equivalent to linear programming [47]. This means that they can be solved exactly in polynomial time. Although an optimal policy for a matrix game can be stochastic, the probabilities and values are guaranteed to be rational if the transitions, rewards, and discount factor are rational.

5.4.2 Iterative Algorithms

From Lemma 5.1 and Theorem 3.2, the value-iteration algorithm can be used to find ϵ-optimal value functions for Markov games

The convergence of policy iteration follows from the convergence of the policy-iteration algorithm for generalized MDPs. Although policy iteration will not necessarily find the optimal value function in finite time, each iteration is guaranteed to improve the current approximate value function by a factor of β. As a result, ϵ-optimal approximations to the optimal policy for a game can be found in time polynomial in the size of the game, $1/(1-\beta)$ and $\log \epsilon$.

5.4.3 Linear Programming

Markov decision processes are Markov games in which the opponent has only one choice of action, and matrix games are Markov games with only one state. Both of these models can be solved exactly in polynomial time using linear programming. Given this fact, it is perhaps surprising that no finite-size linear program can express the optimal value function of an arbitrary Markov game [76]. This follows from the fact that linear programs have rational solutions given rational coefficients, while Markov games can have irrational solutions.

5.5 Complexity Results

Markov games can be solved to any desired degree of accuracy using value iteration. However, it is not known if any algorithm can solve Markov games exactly. This is because the optimal value function of a Markov game can consist of irrational numbers, as was shown by Vrieze [169] using a two-state example. A similar example, which uses only deterministic transitions is described in Section E.1; thus, it is the simultaneous-move quality of Markov games that makes this problem very difficult.

On the other hand, polynomial-horizon problems can be solved in polynomial time using value iteration (see Section 5.3). The value functions for finite-horizon problems are guaranteed to consist of rational numbers as long as the immediate rewards, transitions, and discount factor are rational. Both the deterministic and stochastic versions of the problem are P-complete as they include polynomial-horizon alternating Markov games and matrix games as special cases, both of which are P-hard. See Papadimitriou and Tsitsiklis' [116] work on MDPs for related results.

5.6 Reinforcement Learning in Markov Games

In the reinforcement-learning community, Markov games with simultaneous actions have not been examined as closely as alternating Markov games. This is probably, in part, due to the fact that most popular games are designed for humans to play, and simultaneous actions are cumbersome for humans to carry out.

There are a few examples of learning in simultaneous-action games, including my earlier work on a simple soccer-like game [90], and Harmon, Baird, and Klopf's explorations of a differential pursuit/evasion game [61]. In Harmon et al.'s work, it is assumed that a deterministic optimal policy exists for the simultaneous-action game, whereas, in my work, a stochastic optimal policy is sought.

5.6.1 Minimax-Q Learning

Section 4.6.1 described a Q-learning-like rule for alternating Markov games. It is straightforward to apply the same technique to solving Markov games. An experience tuple is now $\langle s, a_1, a_2, r, s' \rangle$, thus both players must have access to the other's action choice after it is issued. The update is exactly the same as for Q-learning, with the obvious difference that Q values are indexed by the action choices for both players:

$$Q[s, a_1, a_2] := (1 - \alpha)Q[s, a_1, a_2] + \alpha(r + \gamma V(s')),$$

where

$$V(s') = \max_{\rho \in \Pi(\mathcal{A}_1)} \min_{a_2 \in \mathcal{A}_2} \sum_{a_1 \in \mathcal{A}_1} Q[s', a_1, a_2]\rho[a_1].$$

Because the computation of $V(s')$ from the current Q values involves solving a matrix game, each learning step requires solving a linear program.

The algorithm is called minimax-Q because it is essentially identical to the standard Q-learning algorithm with a minimax replacing the maximization. It is described in an earlier paper [90], which includes empirical results on a simple Markov game.

The convergence of this approach follows from the convergence of the generalized Q-learning algorithm in Section 3.6.3. It is interesting to note that, from a convergence-of-learning standpoint, Markov games and MDPs are equally difficult to solve, whereas, from a complexity standpoint, Markov games are significantly harder. This highlights

```
fictitiousMatrix(𝒜₁, 𝒜₂, R, k) := {
    foreach a₁ ∈ 𝒜₁ Y[a₁] := 0
    foreach a₂ ∈ 𝒜₂ X[a₂] := 0
    foreach t ∈ 1...k {
        a₁* := argmax_{a₁∈𝒜₁} Y[a₁]
        a₂* := argmin_{a₂∈𝒜₂} X[a₂]
        foreach a₁ ∈ 𝒜₁ Y[a₁] := Y[a₁] + R(a₁, a₂*)
        foreach a₂ ∈ 𝒜₂ X[a₂] := X[a₂] + R(a₁*, a₂)
    }
    return (max_{a₁∈𝒜₁} Y[a₁])/k
}
```

Table 5.7: Approximating the value of a matrix game by fictitious play.

one of the differences between the criteria used to evaluate learning algorithms and planning algorithms.

5.6.2 Solving Matrix Games by Fictitious Play

Solving a known Markov game using the method of *fictitious play* is reminiscent of reinforcement learning. The basic idea is that we can identify an optimal value function by playing two players against one another. On each step, each player chooses the action that is the best response to the stochastic policy that the other player appears to be using. The long-run proportion of action choices for each player converges to an optimal stochastic policy for the matrix game.

The material in this section is summarized from an article by Vrieze and Tijs [171], which is itself a summary of some 45 years of work in this area. The algorithm in Table 5.7 uses the method of fictitious play to approximate the value of a matrix game.

In this subroutine, k represents the number of rounds of play to use when approximating the game (the method does not necessarily converge in finite time, thus some stopping rule must be used). At step t, vector Y has the property that the value $Y[a_1]$ represents the rewards that the agent would expect to receive in t steps for action a_1 if the actions chosen thus far by the opponent are representative of how it will choose actions in the future. The agent's best (deterministic) response to such an opponent is to choose action $a_1^* = \mathrm{argmax}_{a_1 \in \mathcal{A}_1} Y[a_1]$, the action with the maximum expected reward.

At the same time, the opponent keeps a vector X with one component for each action in $\mathcal{A}_2$. Vector X represents the expected rewards the opponent would receive in t steps when playing against an agent with a fixed stochastic policy in which action a_1 is selected precisely in the proportion in which the agent has chosen it thus far; therefore, $a_2^* = \operatorname{argmin}_{a_2 \in \mathcal{A}_2} X[a_2]$ is the opponent's optimal choice of action.

After actions a_1^* and a_2^* are chosen, the X and Y vectors can be updated to include one more round of rewards. The agent's Y vector is incremented with the rewards the agent receives when the opponent takes action a_2^* (which it just did); similarly for X.

Lemma 5.2 *In the fictitious-play algorithm, the quantities* $(\max_{a_1 \in \mathcal{A}_1} Y[a_1])/k$ *and* $(\min_{a_2 \in \mathcal{A}_2} X[a_1])/k$ *converge to the value of the matrix game, as k increases.*

Proof: This is proven in Vrieze and Tijs' article [171]. □

In addition to the proof, Vrieze and Tijs include information on the rate of convergence of this process, and show that the reward matrix R need not be known with certainty for the process to converge. All that is necessary is for an estimate of R to be available, and for that estimate to converge to R over time.

5.6.3 Solving Markov Games by Fictitious Play

The fictitious-play method for matrix games is interesting from an algorithmic or learning standpoint, but its practical use is extremely limited; solving a game using linear programming is not difficult either conceptually or computationally.

The same is not true of Markov games, which are not known to be exactly solvable by any algorithm. In addition, each phase of the standard iterative methods involve solving a linear program—two, in the case of policy iteration. Therefore, there is much to be said for applying a method like fictitious play to the Markov-game case.

Vrieze and Tijs [171] explored this problem, and found a fictious-play algorithm for Markov games with convergence rates comparable to the matrix game case. Their algorithm is given in Table 5.8.

Although the algorithm is a fairly straightforward extension of the algorithm from the previous section, there are a few subtleties worth explaining. If we knew that V represented the optimal value function for the given Markov game, then the algorithm would essentially be finding the value of the matrix game with payoffs

$$R(s, a_1, a_2) + \beta \sum_{s'} T(s, a_1, a_2, s') V(s')$$

`fictitiousMarkovGame`$(\mathcal{S}, \mathcal{A}_1, \mathcal{A}_2, T, R, \beta, k) := \{$

$\quad M := \max_{s \in \mathcal{S}, a_1 \in {}_1, a_2 \in \mathcal{A}_2} |R(s, a_1, a_2)|$

$\quad$ **foreach** $s \in \mathcal{S}$ $\{$

$\quad\quad$ **foreach** $a_1 \in \mathcal{A}_1$ $Y[s, a_1] := M/(1-\beta)$

$\quad\quad$ **foreach** $a_2 \in \mathcal{A}_2$ $X[s, a_2] := M/(1-\beta)$

$\quad\quad V[s] := \max_{a_1 \in \mathcal{A}_1} Y[s, a_1]$

$\quad \}$

$\quad$ **foreach** $t \in 1 \ldots k$ $\{$

$\quad\quad$ **foreach** $s \in \mathcal{S}$ $\{$

$\quad\quad\quad a_1^* := \mathrm{argmax}_{a_1 \in \mathcal{A}_1} Y[s, a_1]$

$\quad\quad\quad a_2^* := \mathrm{argmin}_{a_2 \in \mathcal{A}_2} X[s, a_2]$

$\quad\quad\quad V[s] := \min(V[s], 1/t\, Y[s, a_1^*])$

$\quad\quad\quad$ **foreach** $a_1 \in \mathcal{A}_1$

$\quad\quad\quad\quad Y[s, a_1] := Y[s, a_1] + R(s, a_1, a_2^*) + \beta \sum_{s'} T(s, a_1, a_2^*, s') V[s']$

$\quad\quad\quad$ **foreach** $a_2 \in \mathcal{A}_2$

$\quad\quad\quad\quad X[s, a_2] := X[s, a_2] + R(s, a_1^*, a_2) + \beta \sum_{s'} T(s, a_1^*, a_2, s') V[s']$

$\quad\quad \}$

$\quad \}$

$\quad$ **return**(V)

$\}$

Table 5.8: Approximating the value of a Markov game by fictitious play.

for each $s \in \mathcal{S}$. However, the value of this game is $V(s)$, so if we knew V, there would be no point in solving this game.

Of course, we do not know the optimal V in advance, but the algorithm in Table 5.8 can be shown to approximate the true optimal value function from above. The algorithm initializes the $Y[s, a_1]$ values with an optimistic estimate of the value of action a_1 in state s. Each time the maximum value of $Y[s, a_1]$ (over all actions) decreases, we can decrease the estimate of the value of state s accordingly.

The fictitious-play algorithm has much in common with a reinforcement-learning approach to this problem; at each step, the players choose actions, and the choice of actions affects the estimates and future decisions. However, it is a very strange learning algorithm. First, for each player to learn to behave optimally, it is necessary for the other player to choose its moves in a particular fashion; the two competing players must collaborate on their choices. Second, updates and action choices are made in all states simultaneously. These two difficulties make the fictious-play approach unsuitable for use in reinforcement-learning problems.

It is likely that the second of these difficulties can be eliminated; the same algorithm ought to converge if Q values are estimated, as in the minimax-Q learning algorithm. On the other hand, it is difficult to imagine eliminating the first difficulty; it seems necessary for the players to choose their actions in this highly scripted way for the fictitious-play approach to converge to optimal behavior.

5.7 Open Problems

In this chapter, I described the problem of solving Markov games, drawing from the fields of algorithmic analysis, reinforcement learning, and game theory. There are a number of extensions that might prove fruitful and interesting.

- Game theorists consider two-player zero-sum games to be the simplest, and in many ways, the least interesting, type of game. Variations that include possible cooperation or multiple players have been considered extensively. Such models are of interest to researchers in planning and reinforcement learning since they could be used to capture interactions between a collection of agents solving tasks together. Are there optimality criteria that would be appropriate for planning and/or learning? Are there efficient algorithms for manipulating these models?

- The use of linear programming in the innermost loop of minimax-Q learning is problematic, because the computational complexity of each step is large and typically many steps will be needed before the system converges sufficiently. Would approximate solutions to the linear programs suffice? The results on fictitious play for Markov games indicate that this ought to be possible. Iterative methods are also quite promising since the relevant linear programs change slowly over time. Are there iterative linear-programming algorithms that would be appropriate for this problem?

- The strength of the minimax criterion is that it allows the agent to converge to a fixed strategy that is guaranteed to be "safe," in that it does as well as possible against the worst possible opponent. It can be argued that this is unnecessary if the agent is allowed to adapt continually to its opponent. To what extent is this true? In theory, any deviation from the minimax-optimal policy would leave the agent vulnerable to a devious form of trickery in which the opponent leads the agent to learn a poor policy and then exploits the resulting situation. Can such an opponent be identified for, say, a regular Q-learning agent?

- The fact that rational-valued Markov games can have irrational value functions makes it hard to discuss the complexity of the optimization problem—how should the algorithm represent and return the irrational values? Decision problems like "Is the optimal value function for state s at least r?" are well-defined (the answer is just one bit), but the exact complexity is unknown. Can it be shown to be uncomputable, perhaps by relating it to the problem of finding the roots of polynomial equations and Galois theory [6]?

- There are criteria other than minimax that capture the competitive aspect of games, while satisfying the conditions for being a generalized MDP. Among these are rules in which agents choose randomly among actions that maximize their worst-case reward and those that maximize their expected reward against a particular adversary. This criterion can be shown to result in well-defined value functions and convergent learning, but are the optimal policies interesting? Do they blend aggressive behavior against a known opponent with conservative actions? Are there other update rules that are more appropriate? Are they non-expansions?

- Markov games can be viewed as incomplete-information games with a particular "information structure" [128] in which the state is made known to both players every other move. It is possible that games with more elaborate information structures can be solved just as efficiently, as long as the structure is not *too* complex. For example, as long as the true state is revealed often enough, it ought to be possible to combine a successive-approximation algorithm with an efficient algorithm for solving game trees [84]. Would such a hybrid algorithm be of interest? Are there any applications with this structure?

5.8 Related Work

Section 4.8 listed work related to alternating Markov games as well as the more general Markov games discussed in this chapter.

Dobkin and Reiss [47] showed that the complexity of solving matrix games is closely related to a set of problems in linear programming and computational geometry; it is interesting to note that their paper was written before linear programming was known to be solvable in polynomial time.

Work on game learning in the reinforcement-learning literature focuses almost exclusively on alternating-move games. Noteworthy exceptions include work by Littman [90] on a discrete soccer-like game, and work by Harmon, Baird, and Klopf [61] on a continuous pursuit-evasion game.

Kallenberg [76] examined a set of game-related problems that can be solved in polynomial time by linear programming. These include one-player games (MDPs), matrix games, and Markov games in which transitions are influenced by only one player. Vrieze [170] surveyed all the algorithms listed in this chapter, as well as others.

General Markov games differ from alternating Markov games in that the players choose actions simultaneously. Games with simultaneous actions can be viewed as a restricted type of incomplete-information game in which the players' actions are issued sequentially but are not revealed until after both players have made their decisions. Koller, Megiddo, and von Stengel [82, 84, 83] looked closely at games of partial information. They developed algorithms that run in polynomial time with respect to the size of the *game tree*, which roughly means that their results apply to Markov games in which there are no cycles in the transition graph. Their algorithms find optimal stochastic policies for a wide range of incomplete-information games including those with simultaneous actions and games that do not obey the zero-sum property.

5.9 Contributions

In this chapter, I described Markov games, a model of sequential decision making in which two agents choose actions in parallel. I discussed several of the classic algorithms for finding approximately optimal solutions to this type of game, and explained that the existence of simple games with irrational solutions makes it difficult to analyze the exact computational complexity of this model. I showed for the first time that this difficulty persists even in deterministic games. In spite of these computational challenges, I presented a novel result that reinforcement-learning algorithms converge to optimal solutions for Markov games.

Because of their computational intractability, Markov games will probably continue to be of mainly theoretical interest to researchers in the fields of reinforcement learning and operations research. However, they could potentially help provide worst-case bounds for even more difficult problems that arise in models with uncertainty in state estimation [123] or imprecise value functions [112]. Even if Markov games themselves are of only marginal interest, the optimal randomness that results from solving them is an important and powerful concept that deserves further attention.

Chapter 6

Partially Observable Markov Decision Processes

Portions of this chapter and its associated appendix have appeared in earlier papers: "Planning and acting in partially observable stochastic domains" [73] with Kaelbling and Cassandra, "Acting optimally in partially observable stochastic domains" [32] with Cassandra and Kaelbling, and "An introduction to reinforcement learning" [74] with Kaelbling and Moore.

Chapter 2 began with an example of a robot deciding how to navigate in a large office building. This hypothetical robot was plagued by an environment that it could not completely control. In spite of these difficulties, I explained how such a robot could use a map of its environment and knowledge of its own dynamics to generate optimal policies for navigating. A more realistic robot not only has unreliable actions, but unreliable observations as well: sometimes a corridor looks like a corner; sometimes a T-junction looks like an L-junction. The MDP algorithms discussed in Chapter 2 are no longer appropriate for an agent that does not have perfect state information.

A robot with imperfect state information cannot use a policy that only maps true location to a best choice of action. In general, the robot will have to remember something about its history of actions and observations, and use this information, together with its knowledge of the underlying dynamics of the world, to maintain an estimate of its location. Many engineering applications follow this approach, using methods like the Kalman filter [77] to maintain a running estimate of the robot's spatial uncertainty, expressed as a Gaussian probability distribution in Cartesian space. This approach will not do for our robot, though. Its uncertainty may be discrete: it might be almost certain that it is in the north-east corner of either the fourth or the seventh floors, though it might admit some chance that it is on the fifth floor, as well.

The robot must decide what actions to take given an uncertain estimate of its location. In some cases, it might be sufficient for the robot to ignore its uncertainty and take actions that would be appropriate for the most likely location. In other cases, it might be better for the robot to take actions for the purpose of gathering information, such as searching for a landmark or reading signs on the wall. In general, it will take actions that fulfill both purposes simultaneously.

6.1 Introduction

In this chapter, I address the problem of choosing optimal actions in partially observable stochastic domains. Problems like the one described above can be modeled as *partially observable Markov decision processes* (POMDPs). In addition to their applicability to problems of robot navigation, POMDPs are useful for solving problems of factory process control, resource allocation under uncertainty, cost-sensitive testing, and a variety of other complex real-world challenges [109].

One important facet of the POMDP approach is that there is no distinction drawn between actions taken to change the state of the world and actions taken to gain information. This is important because, in general, every action has both types of effect. Stopping to ask questions may delay the agent's arrival at the goal or spend extra energy; moving forward may give the agent information that it is in a dead-end because of the resulting crash.

6.2 Partially Observable Markov Decision Processes

Solving a POMDP involves taking a map or model of the environment which includes state transition information, observation probabilities, and the reward structure, and generating a plan for acting to maximize reward.

6.2.1 Basic Framework

A POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{Z}, O \rangle$ is defined in part by an MDP model: a finite set $\mathcal{S}$ of states, and a finite set $\mathcal{A}$ of actions, a transition function $T : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$, and a

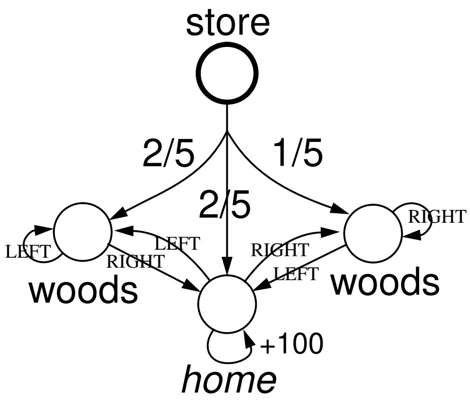

Figure 6.1: An example partially observable environment.

reward function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. In addition, it includes a finite set of observations, $\mathcal{Z}$, and an observation function $O : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{Z})$. The quantity $O(s', a, z)$ is the probability of observing $z \in \mathcal{Z}$ in state s' after taking action a.

6.2.2 Acting Optimally

The average-reward criterion is not always well-defined for POMDPs.[1] This is roughly because, in some problems, the agent can guarantee itself a huge reward tomorrow by doing nothing today, so it ends up doing nothing forever. This is sometimes called the problem of the *infinitely delayed splurge* [121]. However, the optimal value function is well-defined in the discounted case, which I will continue to focus on exclusively here.

Even though the optimal discounted infinite-horizon value function is well-defined, representing a policy that can achieve the optimal value function can be quite challenging. This section reviews some approaches for representing policies.

Memoryless Policies The most naive strategy for dealing with partial observability is to ignore it; that is, to treat observations as if they were the states of the environment and to try to find good behavior. Figure 6.1 shows a simple domain in which the agent is attempting to get "home" from the store. After leaving the store, there is a good chance that the agent will end up in one of two places that look like "woods", but that require different actions for getting home. If we consider these states to be the same, then the agent cannot possibly behave optimally. But how well can it do?

[1] This is also true for MDPs with infinite state or action spaces.

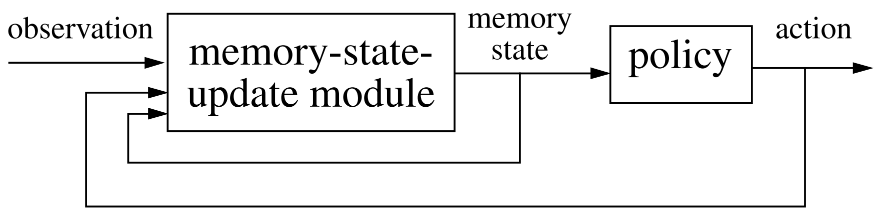

Figure 6.2: Generic structure of memory-based solutions to POMDPs.

Problems relating to finding observation-to-state mappings in POMDPs, sometimes called *memoryless policies*, have been studied in many different contexts [182, 91, 146, 70]. Finding the optimal memoryless policy is NP-hard [91], and it often has very poor performance. In the case of the environment of Figure 6.1, for example, no memoryless policy takes less than an infinite number of steps to goal, on average.

Some improvement can be gained by considering stochastic memoryless policies; these are mappings from observations to probability distributions over actions. If there is randomness in the agent's actions, it will not get stuck in the woods forever. Although algorithms exist for finding locally optimal stochastic policies, finding a globally optimal policy is still NP-hard—this follows indirectly from a result by Papadimitriou and Tsitsiklis [116]. In our woods example, the unique optimal stochastic policy is for the agent, when in the woods, to go RIGHT with probability $2 - \sqrt{2} \approx 0.6$ and LEFT with probability $\sqrt{2} - 1 \approx 0.4$. This can be found by solving a simple (in this case) quadratic program. The fact that the optimal policy requires irrational numbers, even for such a simple example, gives some indication that it is a difficult problem to solve exactly.

Memory-based Policies The only way to behave effectively in a wide-range of environments is to use memory of previous actions and observations to create a better estimate of the current state. There are a variety of approaches to learning policies with memory.

Figure 6.2 illustrates the basic structure. The component on the left is the *memory-state-update module*, which computes the agent's new *memory state* as a function of the agent's present memory state, the most recent action, and the current observation. Solution methods differ in their choice of memory state.

History-window Policies One type of memory-based policy is obtained by defining the agent's memory state to be a list of the k most recent actions and observations.

White and Scherer [178] explored algorithms for finding near-optimal policies of this form. Platzman [121] used a more sophisticated approach that involved variable-width windows: the amount of history stored at any given time depended on the saliency of the most recent actions and observations.

The variable-width-window approach can make much more efficient use of a finite memory because the number of distinct memory states needed for a fixed window of width k is $(|\mathcal{A}||\mathcal{Z}|)^k$, whereas a variable-width-window approach can involve as few or as many memory states as needed.

Finite-memory Policies In the *finite-memory* approach, the memory state can be any one of a finite number of internal states. Finite-memory policies can remember a finite amount of information about the past; unlike history-window approaches, information can be stored for an arbitrarily long time. Both memoryless policies and history-window policies are special cases of finite-memory policies.

Because they are more expressive, general finite-memory policies can be defined that perform better than any history-window policy. However, this additional expressiveness makes optimal finite-memory policies difficult to find. A heuristic algorithm for finding stochastic finite-memory policies has been explored [123]. The value-iteration approach described in Chapter 7 can produce optimal finite-memory policies for some problems [32].

Information States There are POMDPs for which no finite memory is sufficient to define optimal behavior. In contrast, using the memory state to encode every action and observation the agent ever encountered would be sufficient to allow optimal decisions to be made. In many cases, an equivalent yet more convenient representation of the past is the *information state*.

An information state is a probability distribution over states of the environment indicating the likelihood, given the agent's past experience, that the environment is actually in each of those states. A memory-state-update module for information states can be constructed straightforwardly using the environment model and basic probability theory (see Section 7.2.1).

The problem of finding a policy mapping information states into actions can be formulated as an MDP, but the MDP cannot be solved using the techniques of Chapter 2 because the state space is infinite. Chapter 7 addresses techniques for solving this information-state MDP.

6.3 Algorithms for Solving POMDPs

There are no practical algorithms for solving POMDPs. In Chapter 7, I present the witness algorithm, which is the most efficient algorithm to date for solving POMDPs exactly over a finite horizon; it solves some POMDPs very quickly and others quite slowly. Here, I describe several algorithms that are of theoretical interest because they help indicate the computational complexity of different types of POMDPs.

The fundamental decision problem is the following: given a POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{Z}, O, \beta \rangle$, a distribution over starting states x_0, and a reward bound r, is there a policy such that the expected discounted reward starting from x_0 is at least r? I will assume that all the numbers involved are specified as rational numbers written with no more than B bits. I use "POMDP" to refer to the model $\mathcal{M}$, and "POMDP problem" to refer to $\mathcal{M}$ combined with x_0 and the reward bound r.

The complexity of the general infinite-horizon version of this problem is not known. It may be the case that the problem is undecidable, although attempts to prove this have not been successful. On the other hand, the more important problem of finding an ϵ-optimal policy for a given POMDP problem can be solved. I address this problem in more detail in Chapter 7.

6.3.1 Complexity Summary

In this chapter, I present results pertaining to the computational complexity of solving 24 separate variations of the POMDP problem. In this section, I describe upper bounds, in Section 6.5 lower bounds. Here, I will briefly summarize the results.

The basic POMDP problem is, given a POMDP model, an initial distribution x_0, and a reward bound r, is there a policy with value at least r starting from x_0? There are 4 dimensions along which the basic problem might be varied. For each dimension, I will list the values it can take, abbreviations for the values to simplify later discussion, and the relationship between the various values.

- Transitions (T): In general, taking action a from state s results in a stochastic transition. A simpler case is when all transitions and observations are deterministic. Any algorithm for solving stochastic (S) problems can be used to solve deterministic (D) problems by setting the probabilities to zeros and ones; any hardness result for a deterministic problem applies to the stochastic version as well.

- Horizon (t): I am most interested in problems with an infinite (∞) planning horizon. Sometimes, however, computing answers for long enough finite horizons (F) is sufficient. For analytic purposes, it is useful to consider problems in which the horizon length is bounded by a polynomial in the size of the POMDP. Polynomial horizons (P) are a special case of finite horizons. We can create an equivalent infinite-horizon problem for a given polynomial-horizon one by replicating the states once for each time step in the horizon, then adding a zero-reward absorbing state at the end.

- Rewards (R): I will consider general rewards (G), and a special case in which all rewards are non-positive and the target bound is zero. In problems of the latter type, as soon as a single negative reward is encountered with non-zero probability, the value of the policy is less than zero and the reward bound is not met. For this reason, these problems can be formalized using boolean reward values (true for zero rewards, false for negative rewards), so are called boolean-reward problems (B). Discount factors are not needed in the specification of boolean-reward problems.

- Observations (O): When a POMDP has only one possible observation, it is unobservable (U). If there are one or more possible observations, I call it the general case (G).

Table 6.1 summarizes the complexity results presented in this chapter. The parenthesized numbers are section references and "*" represents a "don't care" symbol.

6.3.2 Deterministic POMDPs

A deterministic POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, N, R, \mathcal{Z}, \text{Obs}, \beta \rangle$, is like a general POMDP with the exception that the transition function $N : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and observation function $\text{Obs} : \mathcal{S} \times \mathcal{A} \to \mathcal{Z}$ are deterministic. In this section, I show how to exploit the structure of deterministic POMDP problems to solve them.

The following lemma provides a powerful way to reason about deterministic POMDP problems.

T	t	R	O	
S	∞	G	*	EXPTIME-hard (6.5.1), not known to be decidable
*	F	*	*	decidable (6.3.3)
D	∞	G	*	in EXPTIME (6.3.2), PSPACE-hard (6.5.3)
S	∞	B	G	EXPTIME-comp.: EXPTIME-hard (6.5.1), in EXPTIME (6.3.3)
S	∞	B	U	PSPACE-complete: PSPACE-hard (6.5.3), in PSPACE (6.3.3)
S	P	*	G	PSPACE-complete: PSPACE-hard (6.5.2), in PSPACE (6.3.3)
D	∞	B	*	PSPACE-complete: PSPACE-hard (6.5.3), in PSPACE (6.3.2)
S	P	*	U	NP-complete: NP-hard (6.5.4), in NP (6.3.3)
D	P	*	*	NP-complete: NP-hard (6.5.4), in NP (6.3.2)

Table 6.1: Summary of POMDP complexity results in this chapter.

Lemma 6.1 *For every infinite-horizon deterministic* POMDP *problem* $\langle \mathcal{S}, \mathcal{A}, N, R, \mathcal{Z}, \text{Obs}, \beta, x_0, r\rangle$, *there is a finite-state* MDP *with an equivalent optimal value. The number of states in the* MDP *is no more than* $(1 + |\mathcal{S}|)^{|\mathcal{S}|}$.

Proof: We will construct a finite-state Markov decision process $\langle \mathcal{D}, \mathcal{A}, T', R', \beta\rangle$ that is equivalent to the given deterministic POMDP problem.

In the initial distribution, the probability that the agent is in state s is $x_0[s]$. If the agent is actually in state s, then after taking action a and observing z, the agent is in state $N(s, a)$, assuming that $\text{Obs}(N(s, a)) = z$. If $\text{Obs}(N(s, a)) \neq z$, then we can conclude that it was not possible for the agent to have been in state s initially.

This argument can be generalized to a sequence of actions and observations. For each $t \geq 1$, let a_t be the tth action and z_t be the resulting observation. For each $t \geq 0$, let $D_t : \mathcal{S} \to (\mathcal{S} \cup \{gone\})$. I will refer to D_t as a *table*, and define the value $D_t(s)$ to be the location of the agent at time t, assuming it started in state s. If $D_t(s) = gone$, it was not possible for the agent to have started in state s, given the actions and observations up to time t. Table D_t is defined recursively by: $D_0(s) = s$ for all $s \in \mathcal{S}$ and

$$D_t(s) = \begin{cases} gone, & \text{if } D_{t-1}(s) = gone \\ & \text{or } \text{Obs}(N(D_{t-1}(s), a_t), a_t) \neq z_t; \\ N(D_{t-1}(s), a_t), & \text{otherwise.} \end{cases}$$

We can use elementary probability theory to express the probability that the agent is in state s after t steps in terms of D_t,

$$\Pr(s_t = s) = x_t[s] = \frac{\sum_{s'} x_0[s'] I\{D_t(s') = s\}}{(\sum_{s'} x_0[s'] I\{D_t(s') \neq gone\})}, \tag{6.1}$$

where $I\{e\}$ has value 1 if the boolean expression e is true and zero otherwise. Equation 6.1 simply sums up, over the initial states s' for which the agent would now be in state s, the probability that the agent started in state s', and then normalizes the result. The vector x_t of probabilities is an information state, which is an adequate summary of the past to allow optimal decisions to be made [5].

Since x_t can be written entirely in terms of x_0, which does not change from step to step, and D_t which does, we can use the table D_t to represent the state of the system at time t. As there are $(1+|\mathcal{S}|)^{|\mathcal{S}|}$ possible tables, the state space for a deterministic POMDP with a known initial distribution is finite (in fact, exponential). In the following, $\mathcal{D}$ represents the set of all tables.

The transitions and rewards over the state space of tables are defined as follows. Let N' be a next-table function given a table, action, and observation, $N'(D,a,z) = D'$ where

$$D'(s) = \begin{cases} \mathit{gone}, & \text{if } D(s) = \mathit{gone} \\ & \text{or } \mathrm{Obs}(N(D(s),a),a) \neq z \\ N(D(s),a), & \text{otherwise.} \end{cases}$$

The probability of observing z after taking action a from table D can be found by probability theory to be

$$Pr(z|D,a) = \frac{\sum_s x_0[s] I\{\mathrm{Obs}(N(D(s),a),a) = z\}}{\sum_{z'}(\sum_s x_0[s] I\{\mathrm{Obs}(N(D(s),a),a) = z'\})}.$$

Intuitively, this expression considers each state s and includes its initial probability $x_0[s]$ in the total if the state that it maps to under table D followed by action s results in observation z. It then normalizes this quantity so that it sums to one when all possible observations are considered.

The probability of a transition from D to D' under action a is the sum over observations z of the probability of observing z, given that D goes to D' under observation z: $T'(D,a,D') = \sum_z \Pr(z|a,D) I\{D' = N'(D,a,z)\}$. The expected reward for action a from table D is the sum over states of the reward from state $D(s)$ weighted by the probability that s was the initial state, $R'(D,a) = \sum_s I\{D(s) \neq \mathit{gone}\} x_0[s] R(D(s),a)$.

The Markov decision process $\langle \mathcal{D}, \mathcal{A}, T', R', \beta \rangle$ is equivalent to the given deterministic POMDP problem because at each step, the set of tables and x_0 constitute a sufficient summary of past history. □

Lemma 6.1 shows that the state space of a deterministic POMDP problem possesses a great deal of structure. I will next show that the transitions between states are also constrained in a useful way.

Recall from the proof of Lemma 6.1 that we can represent the state of a deterministic POMDP problem at any moment in time by a table $D : \mathcal{S} \rightarrow (\mathcal{S} \cup \{gone\})$. Define $ng(D) = \sum_s I\{D(s) \neq gone\}$, where I once again is a zero-one indicator function on the given predicate. The quantity $ng(D)$ represents the number of "non-*gone*" elements in the D table. The set of possible next tables for D given action a is a singleton (i.e., the transition is deterministic) if and only if $ng(D') = ng(D)$, where D' is the resulting table. If there are multiple possible next tables, then the value of ng for each of the next tables is strictly smaller than $ng(D)$. This follows from the lemma below.

Lemma 6.2 *Given a table D, and an action a, $ng(D) = \sum_z ng(N'(D, a, z))$.*

Proof: Using the definitions of ng and N', and letting D' be the result of applying N' to D,

$$\begin{aligned}
\sum_z ng(N'(D, a, z)) &= \sum_z \sum_s I\{D'(s) \neq gone\} \\
&= \sum_s \sum_z I\{D(s) \neq gone \text{ and } \mathrm{Obs}(N(D(s), a), a) = z\} \\
&= \sum_s I\{D(s) \neq gone\} \\
&= ng(D).
\end{aligned}$$

□

Lemma 6.2 will help prove two important results concerning deterministic POMDP problems later in this chapter.

Infinite Horizon

Lemma 6.1 leads directly to an algorithm for solving deterministic POMDP problems: create the finite-state MDP described in Lemma 6.1, compute the optimal value function V^* using the linear-programming algorithm of Chapter 2, and then check if $V^*(D_0) \geq r$ (where $D_0(s) = s$ for all $s \in \mathcal{S}$). The run time is exponential in $|\mathcal{S}|$; it is in EXPTIME.

The algorithm and analysis can be improved for special cases. In the boolean-reward case, I will show how to reduce the space requirements to be polynomial, and in the the polynomial-horizon case, I will show how to solve the problem non-deterministically in polynomial time. These results are presented next.

Polynomial Horizon

The optimal t-horizon policy for a POMDP problem can be represented by a *t-step policy tree* (policy trees are discussed in detail in Chapter 7, see Figure 7.3 for a useful illustration). A t-step policy tree is a depth t tree that gives the action choice for the initial state at the root and a $(t-1)$-step policy tree for each observation that is possible given the initial action. A t-step policy tree tells the agent which action to take first and how to behave depending on the observation that results.

In general, a t-step policy tree will have as many as $|\mathcal{Z}|^t$ nodes. In a deterministic POMDP problem, this upper bound can be lowered considerably.

Lemma 6.3 *The optimal t-step policy tree for a deterministic* POMDP *problem has no more than $t|\mathcal{S}|$ nodes.*

Proof: The lemma follows from several useful facts. First, each node in a policy tree can be associated with a table. In particular, the root node is associated with the initial table D_0, and the observation z child of a node with action choice a and table D is associated with table $N'(D, a, z)$.

Second, the optimal policy tree need not include any node whose associated table D would have $ng(D) = 0$. Such a node could never be reached (all observations would be impossible) and so is irrelevant to the representation of the optimal policy.

Third, the sum of the ng values over all the tables at any given level of the policy tree is exactly $|\mathcal{S}|$. This follows from the fact that $ng(D_0) = |\mathcal{S}|$, and Lemma 6.2, which says that the sum of ng values is preserved between a node and its children.

Putting these three facts together, there can be no more than $|\mathcal{S}|$ nodes at any of the t levels of the optimal t-step policy tree, from which the lemma follows. $\square$

Lemma 6.3 shows that, for a polynomial horizon, the number of nodes in the optimal policy tree is polynomially bounded. A polynomial-size policy tree can be evaluated in polynomial time quite easily. For a leaf, let D be the associated table and a be the action chosen; the value of the leaf is $R(D, a)$. For an internal node of the tree, let D be the associated table and a be the action chosen at that node. The value of the node is

$$R(D, a) + \beta \sum_z \text{ value of the } z \text{ child } .$$

We can now specify a polynomial-time non-deterministic (NP) algorithm for solving polynomial-horizon deterministic POMDP problems.

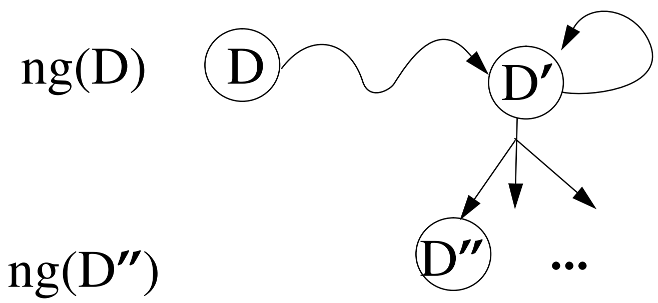

Figure 6.3: Optimal infinite-horizon policy for a deterministic POMDP.

1. Guess the optimal policy tree. By Lemma 6.3, the tree need not have more than a polynomial number of nodes.

2. Evaluate the guessed tree in polynomial time.

3. Return "true" if the value of the root node is at least r.

Boolean Rewards

In the previous section, I showed how to exploit the special structure of the table transition function to show that the optimal polynomial-horizon policy could be represented succinctly. A similar observation can be made for infinite-horizon policies, which makes it possible to solve boolean-reward problems using polynomial space. Recall that in a boolean-reward POMDP problem, all rewards are non-positive and the reward bound is zero.

As a consequence of Lemma 6.2, any sequence of tables followed starting from D_0 can have no more than $|\mathcal{S}|$ stochastic transitions in it. Let D be some table that is reachable from D_0 by following a particular optimal policy. What tables can be visited after D? The agent will take some number of deterministic transitions (no more than $|\mathcal{D}|$), and reach a table D' such that $ng(D') = ng(D)$—all the tables encountered along this path will have ng values equal to $ng(D)$ also. Then, either a loop will be entered (a path of length no more than $|\mathcal{D}|$ from D' to itself), or a stochastic transition will occur to table D'' such that $ng(D'') < ng(D)$. Figure 6.3 depicts the structure of an infinite-horizon policy starting from table D.

In the boolean-reward version of the deterministic infinite-horizon POMDP problem, we want to know whether there is a policy with expected reward equal to zero starting from x_0 (table D_0) given that all immediate rewards are either zero or negative. Let

$V^{0?}(D)$ be a boolean variable indicating whether or not a zero-reward policy exists for an agent starting at table D. The Bellman equation for $V^{0?}$ can be written

$$V^{0?}(D) = \exists a : (R'(D,a) = 0 \wedge \forall z : V^{0?}(N'(D,a,z))).$$

This is not a generalized MDP because boolean arithmetic is being used in place of operations on real numbers.

Using the insight illustrated in Figure 6.3, we can rewrite the Bellman equation in a computationally more convenient form. Let $zpath(D, D', t)$ be a predicate that is true if there is some zero-reward deterministic path from D to D' of length t, and $stoch(D, a)$ be a predicate that is true if there is more than one possible next table resulting from taking action a from table D. The Bellman equation can now be written

$$\begin{aligned} V^{0?}(D) \quad = \quad & \exists D' : \exists 0 \le t \le |\mathcal{D}| : (\; ng(D') = ng(D) \wedge zpath(D, D', t) \wedge \\ & ((\exists 0 \le t' \le |\mathcal{D}| : zpath(D', D', t')) \\ & \vee(\exists a : stoch(D', a) \wedge \forall z : V^{0?}(N'(D', a, z))))\;). \end{aligned}$$

Although this formulation is complicated, it has several important properties. First, $V^{0?}(D)$ is not defined in terms of itself. Although the definition of $V^{0?}$ is recursive, $V^{0?}(D)$ is only defined in terms of $V^{0?}(D'')$ such that $ng(D'') < ng(D)$. Second, close examination of the formula reveals that it can be evaluated using a polynomial amount of space, as long as $zpath$ can be evaluated in a polynomial amount of space.

To see that $zpath$ can be evaluated in a polynomial amount of space, notice that it can be expressed as

$$zpath(D, D', t) = \begin{cases} \text{false}, & \text{if } t = 0 \text{ and } D \ne D', \\ \text{false}, & \text{if } ng(D) \ne ng(D'), \\ \text{false}, & \text{if } t = 1 \text{ and there is no } a \text{ such that} \\ & \quad T'(D, a, D') = 1 \text{ and } R'(D, a) = 0, \\ \text{false}, & \text{if } t > 1 \text{ and there is no } D'' \text{ such that} \\ & \quad zpath(D, D'', \lceil t/2 \rceil) \text{ and} \\ & \quad zpath(D'', D', \lfloor t/2 \rfloor), \\ \text{true}, & \text{otherwise.} \end{cases}$$

It is not hard to see that the above expression for $zpath$ is correct; that it can be evaluated in polynomial space follows from Savitch's Theorem [115].

Since $V^{0?}(D_0)$ is true if and only if the deterministic infinite-horizon POMDP has a zero-reward policy, and $V^{0?}(D_0)$ can be evaluated in polynomial space, the problem of solving boolean-reward POMDPs is in PSPACE.

6.3.3 Stochastic Transitions

When a POMDP has stochastic transitions, the set of information states reachable from a given starting distribution can be countably infinite; this is true even when actions are chosen according to an optimal stationary policy. As a result, although the optimal value function can be approximated to any degree of accuracy in finite time (see Chapter 7), determining whether an infinite-horizon policy can achieve an expected reward of at least r is quite difficult, and perhaps even impossible.

There are special cases of the problem that are decidable. I will next show that when reward is restricted to be non-positive, the existence of a zero-reward optimal policy can be determined using an exponential amount of time. I will also show that when the horizon is restricted to be finite, the problem is decidable, and if the horizon is polynomially bounded, the problem can be decided in a polynomial amount of space.

Polynomial Horizon

An information state is a probability distribution over the states in $\mathcal{S}$. In the next chapter, I will explain how an information state can be updated to summarize new information, in the form of actions and observations. For the purposes of this section, all we need to know is that information states are sufficient summaries of past history for predicting future transitions and rewards, and that the set of information states reachable in a finite number of steps is finite.

The t-step value of information state x can be written

$$V_t(x) = \max_a \left(\sum_s x[s]R(s,a) + \beta \sum_z \Pr(z|x,a)V_{t-1}(x') \right),$$

where $V_0(x) = 0$ and x' is the information state resulting from taking action a and observing z from information state x. It is straightforward to evaluate this expression in finite time for $t < \infty$ and in polynomial space if t is polynomially bounded [116]. The resulting optimal value can be compared to the reward bound r to answer the decision problem for a finite-horizon POMDP problem.

In the case of an unobservable POMDP over a polynomial horizon, the optimal policy is a polynomial-length sequence of actions; such a policy can be guessed and evaluated in polynomial time, therefore the associated POMDP problem is in NP.

Boolean Rewards

To compute whether a given policy achieves zero total reward, given that immediate rewards are all non-positive, it is not necessary to keep accurate statistics about the agent's information state. For information state x, if $x[s] > 0$ and taking action a from state s results in a negative reward, then taking action a from information state x results in negative reward.

It is sufficient, therefore, to group information states by the set of states to which they assign positive probability. Using this insight, the total number of distinct groups of information states is $2^{|\mathcal{S}|} - 1$, the size of the power set of $\mathcal{S}$ minus the null set. For the boolean-reward case, it is possible to define a boolean-reward MDP with these groups as the states. This MDP has a zero-reward optimal policy if and only if the boolean-reward POMDP has one, and can be solved in exponential time using the linear-programming algorithm of Chapter 2, or by a simple graph search algorithm.

In the case of an unobservable POMDP, the finite-state boolean-reward MDP described above is deterministic and can be solved using polynomial space using a variation of the *zpath* predicate from Section 6.3.2.

6.4 Algorithmic Analysis

The presentation of the algorithms in Section 6.3 included analyses of their upper bounds—it is worth noting that these algorithms can all be implemented to run in exponential time because all NP, PSPACE, and EXPTIME algorithms can. Therefore, the worst-case run times (and most of the best-case run times) for the algorithms in the previous section are exponential.

6.5 Complexity Results

In this section, I collect what is known of the complexity of solving POMDPs. I show that

- the infinite-horizon problem is EXPTIME-hard and EXPTIME-complete in the case of boolean rewards;
- the polynomial-horizon problem is PSPACE-complete [116], even in the boolean-reward case;
- the infinite-horizon, deterministic case (observable or not) is PSPACE-hard, and PSPACE-complete in the boolean-reward case; and
- the polynomial-horizon, deterministic problem is NP-complete, even in the boolean reward, unobservable case.

Each result is a lower bound, stated in its most specific form. It is important to keep in mind that hardness results for boolean-reward models also apply to general-reward models, and hardness results for unobservable models also apply to partially observable models. I will summarize the implications of these results in Section 6.5.5.

6.5.1 Infinite Horizon

The infinite-horizon, boolean-reward POMDP problem is: Given a POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{Z}, O \rangle$ in which all rewards are non-positive, and a set of non-zero probability initial states S_0, is there a policy that achieves zero reward over the infinite horizon starting from every state in S_0?

This problem is provably intractable. In Section F.2, I show that the problem is hard for EXPTIME, which implies that any algorithm for solving it must take exponential time for infinitely many instances. See Papadimitriou's complexity book [115] for background information on the class EXPTIME.

6.5.2 Polynomial Horizon

The polynomial-horizon, boolean-reward POMDP problem is: Given a POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{Z}, O \rangle$ in which all rewards are non-positive, a polynomially bounded horizon length t, and a set of non-zero probability initial states S_0, is there a policy that achieves zero reward over t steps starting from every state in S_0?

A polynomial-time algorithm for solving this problem could be used to solve quantified-boolean-formula problems in polynomial time. Since the quantified-boolean-formula problem is PSPACE-hard [55], this shows that the polynomial-horizon, boolean-reward POMDP problem is also PSPACE-hard. The proof is due to Papadimitriou and

Tsitsiklis [116].

6.5.3 Infinite Horizon, Deterministic

The unobservable, deterministic, infinite-horizon, boolean-reward POMDP problem is: Given a deterministic, unobservable POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, N, R \rangle$ in which all rewards are non-positive, and a set of non-zero probability initial states S_0, is there a policy that achieves zero reward over the infinite horizon starting from every state in S_0?

A polynomial-time algorithm for solving this problem could be used to solve finite-state-automata-intersection problems in polynomial time. Since the finite-state-automata-intersection problem is PSPACE-hard [55], this shows that the unobservable, deterministic, infinite-horizon, boolean-reward POMDP problem is also PSPACE-hard. The proof is given in Section F.1.

6.5.4 Polynomial Horizon, Deterministic

The unobservable, deterministic, polynomial-horizon, boolean-reward POMDP problem is: Given a deterministic, unobservable POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, N, R \rangle$ in which all rewards are non-positive, a polynomially bounded horizon-length t, and a set of non-zero probability initial states S_0, is there a policy that achieves zero reward over t steps starting from every state in S_0?

A polynomial-time algorithm for solving this problem could be used to solve 3-CNF-SAT problems in polynomial time. Since the 3-CNF-SAT problem is NP-hard [55], this shows that the unobservable, deterministic, polynomial-horizon, boolean-reward POMDP problem is also NP-hard. The proof is a corollary of the result from Section 6.5.2, due to Papadimitriou and Tsitsiklis [116].

6.5.5 Complexity Summary

Table 6.1 summarized the complexity results (lower bounds) presented in this section, as well as the upper bounds derived from the algorithms in Section 6.3. Figure 6.4 gives a more abstract summary. In the figure, the most general problem is at the top and the most constrained is at the bottom, the prefix "D-" means deterministic, the suffix "-poly" means polynomial horizon, and "UMDP" means unobservable MDP. The exact "complete" complexity class for the boolean-reward problem is given, and when

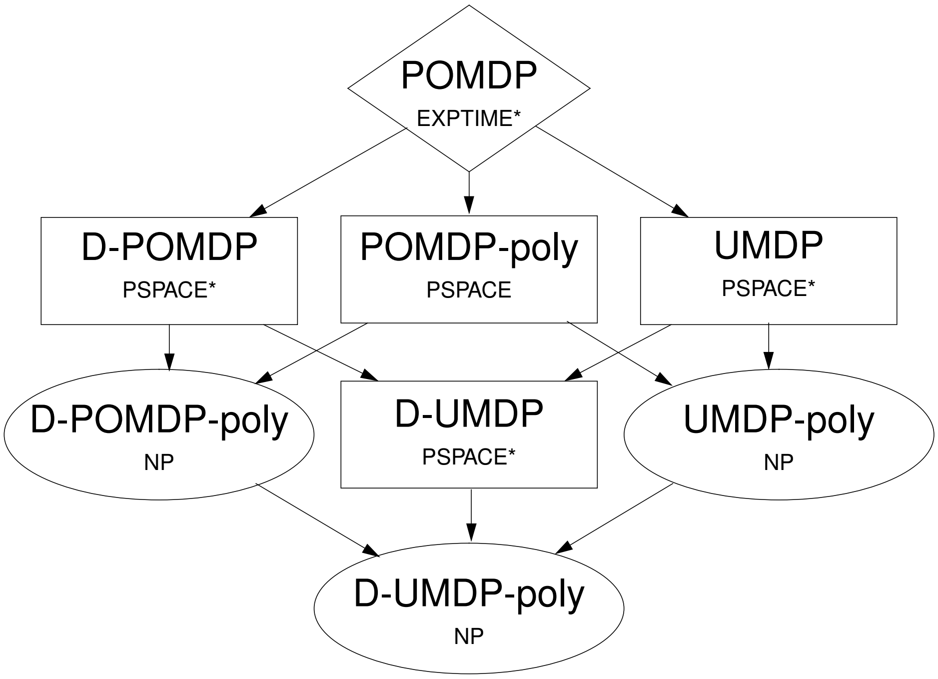

Figure 6.4: An abstract summary of complexity results for POMDPs.

the general-reward problem is not known to be in the same class, it is marked with an asterisk.

There are a few observations worth making about the information in the figure. Although, historically, stochastic models have been viewed as more difficult than their deterministic counterparts, this is not consistently the case. In particular, in unobservable models, restricting the problem to deterministic transitions does not change the complexity class.

A more important simplification is shifting from infinite-horizon problems to polynomial-horizon problems. This consistently improves the complexity (EXPTIME to PSPACE, PSPACE to NP), and makes it possible to solve general-reward problems as easily as boolean-reward problems. This provides additional support to the idea that approximating an infinite-horizon solution by a finite-horizon solution is an efficient approach.

6.6 Reinforcement Learning in POMDPs

Reinforcement learning in partially observable domains is a much more difficult problem than reinforcement learning in completely observable domains, such as MDPs or Markov games. In general, an experience tuple in a POMDP is $\langle a_t, z_t, r_t \rangle$: the most recent action, observation, and reward. Whereas, in completely observable models, it is possible for the reinforcement learner to find the optimal value function when experience tuples are presented in an arbitrary order, in POMDPs, the proper ordering is crucial to learning; previous experience tuples are part of the agent's past history, and therefore its state.

Because of these difficulties, no reinforcement-learning method for general POMDPs is known to converge to an optimal policy. Nonetheless, there are some heuristic methods that are known to do well on simple problems. It is an open problem whether any of these methods can be used to solve realistic problems.

As before, I classify reinforcement-learning methods according to whether or not they learn a model of the environment, or simply try to learn a policy directly. I subclassify model-free approaches by whether any form of short-term memory is used.

6.6.1 Model-free Methods, Memoryless

In the memoryless approach to reinforcement learning in POMDPs, the learner treats observations as if they were states. Two consecutive experience tuples can be combined to create an *observation tuple* of the form $\langle z_{t-1}, a_t, z_t, r_t \rangle$. This tuple is similar to the experience tuple in MDPs, with observations replacing states.

In POMDPs for which the immediate observation completely distinguishes the current state, this observation tuple is sufficient for learning optimal behavior (see Section 2.6.1). In other POMDPs, it is not sufficient for learning; even if immediate observations are enough to make optimal action choices, learning which choices to make can require additional information about past history [103].

Q-learning Many researchers have used Q-learning and other MDP-based reinforcement-learning algorithms to learn policies for partially observable domains. One interesting example is Wilson's work on a 900-state POMDP [182]. A classifier-based approach, and later Q-learning, were both able to find acceptable memoryless policies for this large domain. It has been shown [91] that neither approach finds the optimal memoryless policy. In addition, there are examples that show that Q-learning can fail to converge or find pessimal policies in partially observable domains; see Section F.3.

Stochastic policies Even if Q-learning were able to find the optimal observation-to-action policy, it would be of little use in general; in many problems, no observation-to-action policy achieves acceptable levels of reward. By broadening the class of memoryless policies to *stochastic memoryless policies*, it is possible to improve the situation somewhat (recall the example in Figure 6.1).

Jaakkola, Singh and Jordan [70] developed a reinforcement-learning algorithm that learns policies that choose actions probabilistically on the basis of the current observation. Their algorithm converges to locally optimal policies, meaning that no local change to the probabilities results in improved performance, although it might be possible to adopt an entirely different policy that does substantially better. There has been extremely little computational experience with this algorithm; it is difficult to judge its usefulness at this time.

The algorithm itself is interesting in its use of TD(1)-type updates. This means that, during learning, there is a great deal of record keeping and statistics gathering. Nonetheless, the method is considered "memoryless" in that the policy that is learned does not require the agent to maintain any memory of the past in deciding which action to choose.

Consistent Representations Whitehead and Lin [179] demonstrate that, for some environments, it is possible to coax a learning algorithm to adopt a good memoryless policy, if one exists. A *consistent representation*, in their framework, is one in which the states visited in the course of executing a policy can be adequately distinguished on the basis of their observations. Whitehead and Lin present an algorithm that is appropriate for POMDPs in which the agent has some degree of control over its observations.

6.6.2 Model-free Methods, Memory-based

Unlike the memoryless methods, memory-based methods construct policies that require the agent to maintain some form of short-term memory during the execution of the policy. Such policies can be significantly better than memoryless policies when the crucial states of the environment cannot be distinguished on the basis of their observations. There are environments for which no finite amount of memory suffices for constructing an optimal policy; it is not yet clear whether important or practical environments have this property.

Suffix tree The suffix-tree approach is closely related to Platzman's variable-width-history-window approach [122], mentioned earlier. McCallum [105] showed how a good policy can be learned in the absence of a model by iteratively widening the history window at points that appear to benefit from additional history information. A closely related technique was explored by Ring [129] from a "neural" perspective.

A recent extension of the suffix-tree approach [103], adapted to deal with large, structured observation spaces, has been applied to a simulated highway-driving task with over 21,000 states, 6,000 observations, and five actions. The learned policy used about 150 internal states, and was able to handle many tricky situations; however, it was, by no means, optimal.

Recurrent Q-learning One intuitively simple approach is to use a recurrent neural network to learn Q values. The network can be trained using backpropagation through time or some other suitable technique, and learns to retain "history features" to predict value. This approach has been studied by a number of researchers [106, 89, 137]. It seems to work effectively on simple problems, but can suffer from convergence to local optima on more complex problems.

Register memory Another short-term memory structure that has been studied in the reinforcement-learning framework is storage registers [72, 91, 181, 35]. The idea here is that the agent has explicit actions for saving information in non-volatile memory, and for retrieving this information at a later time.[2] The method has been used successfully when the number of storage registers is small (one or two), but the combinatorics appear to make this approach impractical when the number of registers is larger.

6.6.3 Model-based Methods

As in the completely-observable case, we can learn to solve a POMDP by breaking the process into two parts: first, learn the POMDP model from experience, then (or concurrently) find an optimal policy for the model. Given a model, a policy can be found

[2]This type of memory can be viewed as a form of *stigmergy* [12]. The idea behind stigmergy is that the actions of an agent change the environment in a way that affects later behavior resulting in a form of "external memory."

using techniques from Chapter 7; both algorithmic methods and learning methods are appropriate. This section describes several attempts at learning the model itself.

Chrisman [34] showed how the Baum-Welsh algorithm [11] for learning hidden Markov models (HMMs) could be adapted to learning transition and observation functions for POMDPs. He, and later McCallum [104], gave heuristic state-splitting rules to attempt to learn the smallest possible model that captures the structure of a given environment.

The Baum-Welsh algorithm is known to converge to locally optimal models, and the same is probably true of its application to learning POMDP models. However, no method is known for converging to a globally optimal model for general POMDPs. Thus, even if an optimal policy could be found for the learned model, there is still no guarantee that the process would converge to an optimal policy for the environment.

In their work on model-based methods for POMDPs, Chrisman and McCallum learned a particular representation of the value function that did not require an explicit representation of the reward function. To apply the sophisticated techniques of Chapter 7 in the context of a learned model, it is necessary to represent the reward function R directly. Fortunately, this is not difficult to do.

Given a learned POMDP, it is possible to use the history of actions and observations to construct an information-state experience tuple, $\langle x_t, a_t, r_t, x_{t+1} \rangle$. We want to find a reward function R that has the property that $\sum_s x_t[s]R(s, a_t)$ is the expected value of r_t. Assuming the information states are properly maintained, this is equivalent to a supervised-learning problem with a simple linear function. The update rule

$$\Delta R(s, a_t) = \alpha_t x_t[s] \left(r_t - \sum_s x_t[s] R(s, a_t) \right),$$

where α_t is a learning rate, can be shown to make the reward function converge to one that predicts the immediate rewards arbitrarily accurately [66].

6.7 Open Problems

The study of algorithmic and complexity properties of POMDPs is still relatively young. Although the results I presented in this chapter constitute significant progress towards understanding these problems, many important issues remain unresolved.

- Given a POMDP, an initial distribution, and a reward bound, is it possible to determine whether there is an infinite-horizon policy that can achieve the reward bound or better from the given initial distribution? What about when the POMDP is unobservable? There is some reason to believe that the problem is undecidable, but proving this appears extremely difficult.[3]

- Is there is a POMDP problem and initial distribution, all represented with rational numbers, whose optimal value from that initial distribution has irrational value? The answer to this might shed some light on the decidability of POMDP problems.

- Consider a POMDP in which there is a zero-reward absorbing state that is reached with probability 1 under all policies. If the POMDP is completely observable, this condition is the all-policies-proper condition, and the optimal value function is bounded even if $\beta = 1$. Is this also true when the POMDP is partially observable? If the minimum probability of reaching the absorbing state is non zero from all states, say p, the answer is yes: an equivalent POMDP can be created by decreasing the probability of reaching the absorbing state and setting the discount factor to $1 - (1 - \beta)p$.

- Several researchers [178, 122] have shown that POMDPs with no zero probabilities in their transition or observation matrices can be solved arbitrarily well by policies that remember a finite amount of history. This is because non-zero probabilities have a tendency to make distant observations and actions irrelevant to current decision making; this can be viewed as *informational* discounting, analogous to the value discounting that makes distant future rewards irrelevant to current decision making. Are POMDPs with informational discounting easier to solve than general POMDPs? Are they decidable? Can the idea of informational discounting make it possible to analyze approximate state estimators?

- Sondik [149] defines the class of finitely transient policies, and shows that these policies can be represented as finite-memory policies. The POMDP decision problem described earlier is decidable for the class of POMDPs with finitely transient optimal policies, because we can simply enumerate all the finite-memory policies until one is found that is provably optimal. Can value iteration be used to identify optimal policies for finitely transient POMDPs? Is informational discounting guaranteed to make a POMDP finitely transient?

[3] Work currently in progress by Hanks uses a result from the probabilistic automata literature [118] to show that problem of solving undiscounted POMDPs is undecidable; it is unclear whether these results can be adapted for discounted POMDPs.

- The problem of finding the optimal value for a finite-horizon deterministic POMDP is NP-complete. It has been shown that there are heuristics that are useful for finding optimal memoryless policies [91], another NP-complete problem. Are there good heuristics for solving finite-horizon deterministic POMDPs?
- The search method used in the PSPACE algorithm for computing optimal values for deterministic, infinite-horizon POMDPs with boolean rewards (Section 6.3.2) also works for general-reward POMDPs, except that the values themselves may require an exponential number of bits to write down. Is there a way to represent the optimal values more compactly? If so, it might be possible to extend the PSPACE result to cover general-reward POMDPs.
- The existence of Bellman equations for the boolean-reward case (Section 6.3.2) suggests that it might be possible to develop a theory of optimal policies for other algebraic structures. Are there other algebraic structures that could be used in a sequential decision-making setting?
- Is there a class of *natural* POMDPs? The carefully constructed hard POMDPs I reference in this chapter do not seem very natural. Do the POMDPs found in real-world problems have structure that makes them any easier to solve?
- The complexity results in this chapter address the difficulty of computing exact solutions to POMDPs. Finding approximate solutions is likely to be easier. What is the complexity of computing approximately optimal POMDP policies?
- Is there a reinforcement-learning method that converges to an optimal policy? Although no such algorithm is known, I believe McCallum's suffix-tree algorithm is close to being convergent, at least applied to POMDPs with optimal finite-history policies. Is there a non-trivial subclass of POMDPs that can be solved by reinforcement learning? Would it help to consider "natural" POMDPs?

6.8 Related Work

In this chapter, I presented partially observable Markov decision processes, gave a brief overview of solution methods that have been employed to solve them, developed associated complexity results, and described reinforcement-learning approaches.

The fundamental mathematical structure of POMDPs was developed by Drake [48] and Aström [5]. The algorithmic foundation was laid by Sondik [149, 150]. Additional information on algorithmic approaches can be found in Section 7.8.

State estimation in a type of continuous-space POMDP was explored by Kalman [77] and others, although the "spatial" assumptions required by Kalman's approach are violated for the graph-like state spaces considered in this chapter.

Several researchers have explored the problem of finding finite-memory policies for POMDPs. Platzman [121] developed a finite-history-window approach in his thesis, and later explored a heuristic method for finding more general stochastic finite-memory policies [123]. White and Scherer [178] also presented bounds on the suboptimality of a type of finite-history-window approach. Cassandra, Kaelbling, and Littman [32] showed how finite-memory policies can sometimes be found using value iteration. Littman [91] showed that finding good memoryless policies is NP-complete; closely related results were proven by Papadimitriou and Tsitsiklis [116] and Koller and Megiddo [82]. Both Littman's and Papadimitriou and Tsitsiklis' proofs can be extended to show that finding optimal stochastic memoryless policies is NP-hard.

The study of the complexity of Markov decision processes was initiated by Papadimitriou and Tsitsiklis [116]. For POMDPs, they showed that the finite-horizon problem is PSPACE-hard. More recent work by Burago, de Rougemont and Slissenko [31] showed that a class of POMDPs with bounded unobservability can be solved in polynomial time. They introduced a parameter m which is a measure of how "unobservable" the environment is; given that observations are a deterministic function of the state, m is the largest number of states that possess the same observation. Special cases where $m = 1$ (completely observable) and $m = |\mathcal{S}|$ (completely unobservable) have been studied separately.

The relevant reinforcement-learning literature is quite varied. Singh, Jaakkola and Jordan [146] presented the theory behind defining optimal memoryless (and by extension, finite-memory) policies in partially observable reinforcement-learning environments. They argued persuasively in favor of using an undiscounted criterion. Jaakkola, Jordan and Singh [70] described a provably convergent reinforcement-learning algorithm for maximizing undiscounted reward in partially observable environments; their algorithm finds locally optimal stochastic memoryless policies.

Wilson [182] presented results on applying a classifier-system-based memoryless reinforcement-learning algorithm to a large partially observable environment to fairly good effect. Littman [91] repeated these experiments using Q-learning with similar results. Wilson [181] recently suggested a register-memory extension to a classifier system; Cliff and Ross [35] implemented this idea and found that it works well for very simple problems. McCallum [105] examined a tree-based-memory approach for simultaneously learning a predictive model and an approximate value function.

Neural networks have been used to find short-term memories for reinforcement-learning agents. Schmidhuber [137] designed a novel connectionist learning algorithm for Markovian and non-Markovian environments. Lin and Mitchell [89] surveyed several different architectures including finite-history windows, observation prediction models, and recurrent networks for approximating the optimal value function. Meeden, McGraw, and Blank [106] applied a simple backpropagation-based algorithm [2] to a recurrent network that learned to drive a remote-controlled car. Lin and Whitehead [179] presented reinforcement-learning algorithms for learning internal representations of the state, and an algorithm for learning to behave in such a way as to obviate the need for internal state representation. In all these papers, the problems described would be difficult to formalize as POMDPs; some involve continuous state spaces, others act in the real-world in the absence of a model, and the rest have state spaces that are so large that it would be difficult to solve them using existing algorithms. However, without additional formal analysis, it is difficult to predict whether these results will scale well to larger domains.

Chrisman [34] and McCallum [104] extended an algorithm for learning hidden Markov models to the case of learning a reward and transition model for a POMDP. Their algorithms used an impoverished linear representation for the value function, but sophisticated rules for determining when to extend the number of states in the approximate model; Chrisman used a rule based on the accurate prediction of observations, and McCallum used a rule based on the accurate prediction of values. Bengio and Frasconi [14] created an algorithm for learning input/output HMMs, a model that is equivalent to a POMDP with no rewards. Abe and Warmuth [1] studied the problem of learning approximately correct probabilistic automata from experience. Their learning framework is very interesting, and worth extending to POMDPs. Hernandez-Lerma and Marcus [65] approach the problem of reinforcement-learning in POMDPs from a different perspective; their results show that *given* a method for learning a parameterized model of the environment, it is possible to use a variation of value iteration to simultaneously learn the model and converge to an optimal policy.

6.9 Contributions

In this chapter, I examine partially observable Markov decision processes. My fundamental contribution to this area is a collection of complexity results that show how difficult it is to select optimal actions for this model, even when the problem is constrained in various ways. I explain that no provably convergent learning algorithm exists and I augment the existing heuristic model-learning algorithms by devising a new algorithm for learning the reward function for an unknown POMDP.

As the complexity results in this chapter show, POMDPs are simply too difficult to solve. However, they are also too important to ignore. Perhaps a resolution of this difficulty will come when researchers begin to explore applications of POMDPs to important real-world problems. The constraints present in these applications might be sufficient to make the corresponding POMDPs solvable. Some progress has been made: Hansen [60] blended completely unobservable and completely observable MDPs to form an intermediate model, and Simmons and Koenig [144] controlled a robot using a POMDP model. From the interest that has been generated, I believe it is likely that a great deal of additional progress will be made in the next few years.

Chapter 7

Information-State Markov Decision Processes

Portions of this chapter and its associated appendix have appeared in earlier papers: "Planning and acting in partially observable stochastic domains" [73] with Kaelbling and Cassandra, "Acting optimally in partially observable stochastic domains" [32] with Cassandra and Kaelbling, "The witness algorithm: Solving partially observable Markov decision processes" [92], "Learning policies for partially observable environments: Scaling up" [94] with Cassandra and Kaelbling, and "An efficient algorithm for dynamic programming in partially observable Markov decision processes" [95] with Cassandra and Kaelbling.

In this chapter, I present a number of algorithms for solving information-state Markov decision processes. As discussed in Section 6.2.2, an information-state MDP arises in the context of solving POMDPs when the agent encodes its history of actions and observations as a probability vector over states of the environment. The algorithms from Chapter 2 are not adequate because the information-state MDP has an infinite number of states—the algorithms in Chapter 2 apply only to finite-state MDPs.

7.1 Introduction

An *information state* is a vector of probabilities, one probability value for each state in the POMDP, that sums to one. Given a POMDP model, information states can be

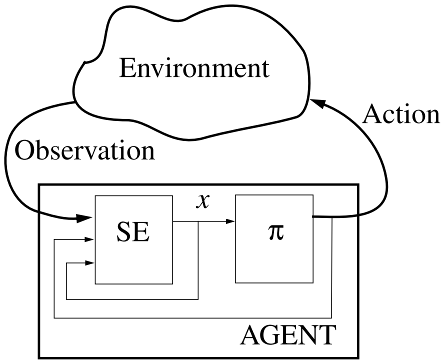

Figure 7.1: A POMDP agent can be decomposed into a state estimator (SE) and a policy (π).

updated using basic probability theory and encode sufficient information for making optimal decisions. Because they constitute a sufficient statistic for optimal behavior, it is possible to use information states to define a particular kind of infinite-state MDP [5].

7.2 Information-state MDPs

Information-state Markov decision processes arise when the problem of controlling a POMDP is decomposed into the two components shown in Figure 7.1. The agent makes observations and generates actions. It uses memory to summarize its previous experience. The component labeled SE in the figure is the *state estimator*: it is responsible for updating the memory state based on the most recent action and observation and the previous memory state (it is a type of memory-state-update module, as discussed in Section 6.2.2). The component labeled π is the policy: as before, it is responsible for generating actions, but now as a function of the agent's memory state rather than the state of the environment.

In this chapter, the contents of the agent's memory is an *information state*: a probability distribution over states of the environment. Information states are sufficient summaries of past history to make optimal decisions. This is because, given the agent's current information state, no additional data about its past actions or observations would supply any further information about the current state of the environment.

Figure 7.2 illustrates a simple POMDP with four states, one of which is a goal state marked with a star. There are two possible observations: one is always made when the agent is in state 1, 2, or 4; the other, when it is in the goal state. There are two

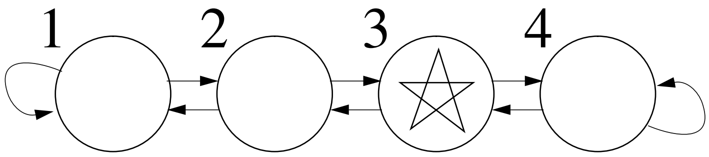

Figure 7.2: A simple POMDP example.

possible actions: RIGHT and LEFT. These actions succeed with probability 0.9, and when they fail the movement is in the opposite direction. If no movement is possible in a particular direction, then the agent remains in the same location.

We assume that the agent is initially equally likely to be in any of the three non-goal states. Thus, its initial information state is $(0.333, 0.333, 0.000, 0.333)$, where the order of components in the vector corresponds to the order of states in the figure.

If the agent takes action RIGHT and does not observe the goal, then the new information state is $(0.100, 0.450, 0.000, 0.450)$. Not observing the goal a second time after taking action RIGHT results in an information state in which the right-most state is most probable: $(0.100, 0.164, 0.000, 0.736)$. Notice that as long as the agent does not observe the goal state, it will always have some non-zero chance that it is in any of the non-goal states; only the third component of the information state will be zero.

7.2.1 Computing Information States

An information state x is a probability distribution over $\mathcal{S}$. We let $x[s]$ denote the probability assigned to state s by information state x. The axioms of probability require that $0 \leq x[s] \leq 1$ for all $s \in \mathcal{S}$ and that $\sum_{s \in \mathcal{S}} x[s] = 1$. The state estimator must compute a new information state x', given an old information state x, an action a, and an observation z. The new probability of some state s', $x'[s']$, can be obtained from basic probability theory as follows:

$$\begin{aligned} x'[s'] &= \Pr(s'|z, a, x) \\ &= \frac{\Pr(z|s', a, x) \Pr(s'|a, x)}{\Pr(z|a, x)} \\ &= \frac{\Pr(z|s', a) \sum_{s \in \mathcal{S}} \Pr(s'|a, x, s) \Pr(s|a, x)}{\Pr(z|a, x)} \\ &= \frac{O(s', a, z) \sum_{s \in \mathcal{S}} T(s, a, s') x[s]}{\Pr(z|a, x)} \end{aligned}$$

The denominator, $\Pr(z|a,x)$, can be treated as a normalizing factor, independent of s', that causes x' to sum to 1. The state-estimation function $\mathrm{SE}(x,a,z)$ has as its output the new information state x'.

Thus, the state-estimation component of a POMDP controller can be constructed quite simply from a given model.

7.2.2 Basic Framework

The policy component in Figure 7.1 takes an information state as input and produces an action. Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{Z}, O, \beta \rangle$ be a partially observable Markov decision process. Because the information state is a sufficient statistic, we can treat it as a state and define the information-state MDP, $\mathcal{B} = \langle \mathcal{X}, \mathcal{A}, N, \tau, \rho \rangle$, where $\mathcal{X}$ is the $|\mathcal{S}-1|$-dimensional unit simplex representing the set of all information states, $N(x,a) = \{\mathrm{SE}(x,a,z) | z \in \mathcal{Z}\}$ is a next-state function for information states, $\tau(x,a,x') = \sum_z I\{\mathrm{SE}(x,a,z) = x'\} \Pr(z|a,x)$ is the information-state transition function, and $\rho(x,a) = \sum_s x[s]R(s,a)$ is the information-state reward function.

This information-state MDP has the property that an optimal policy for it, coupled with the state-estimation function, will give rise to optimal behavior (in the discounted infinite-horizon sense) in the original POMDP [150, 5]. The remaining problem, then, is to solve this MDP. It is very difficult to solve continuous-space MDPs in the general case [133], but, as we shall see in the next section, the information-state MDP has special properties that can be exploited to simplify its solution.

7.2.3 Acting Optimally

The continuous nature of the information-state MDP presents several challenges to finding optimal behavior computationally. As in the case of finite-state MDPs, the target is a policy that maximizes discounted expected reward, and this policy can be defined as the greedy policy with respect to the optimal value function. Once again, the optimal value function is well-defined and can be approximated by value iteration. The primary difficulty is that the value function can no longer be represented by a table of values, one for each state, because the state space itself is continuous.

There appears to be no method for representing general optimal value functions for infinite-horizon information-state MDPs. The best we can hope for is an approximation. In this chapter, I discuss algorithms that address this issue using a parameterized representation of the exact value functions produced in value iteration; algorithms have been developed that attempt to represent approximations of the infinite-horizon value function more directly [100], but I will not discuss these representations here.

7.3 Algorithms for Solving Information-state MDPs

The information-state MDP is a special kind of MDP, and many different algorithms are available for solving it. The algorithms in Chapter 2 do not apply directly, because those algorithms were designed for finite-state MDPs. However, versions of policy iteration [150] and value iteration [135] have been developed for the information-state MDP.

The algorithms I present in this section are all variations of value iteration. They find near-optimal infinite-horizon value functions by exactly solving for t-step finite-horizon value functions for larger and larger t, until the difference between successive value functions is sufficiently small. Section 7.3.1 shows that finite-horizon value functions for the information-state MDP are always piecewise-linear and convex, implying that they can be exactly represented by a finite set of linear functions. This is not necessarily true for the infinite-horizon discounted value function; it remains convex [175], but may have infinitely many facets. I present the algorithms from the simplest and least efficient, to the most complicated and most efficient, including a novel algorithm called the witness algorithm [95].

7.3.1 The Policy-Tree Method

In this section, I present a simple algorithm for finding t-step value functions that, although impractical, serves as the basis for the more efficient algorithms developed in the remainder of the chapter. We begin by considering the structure of optimal finite-horizon policies.

When an agent has one step remaining, all it can do is take a single action. With two steps to go, it can take an action, make an observation, then take another action, perhaps depending on the previous observation. In general, an agent's non-stationary t-step policy can be described by a *policy tree* as shown in Figure 7.3. It is a tree of depth t that specifies a complete t-step policy. The top node determines the first action to be taken. Then, depending on the resulting observation, an arc is followed to a node

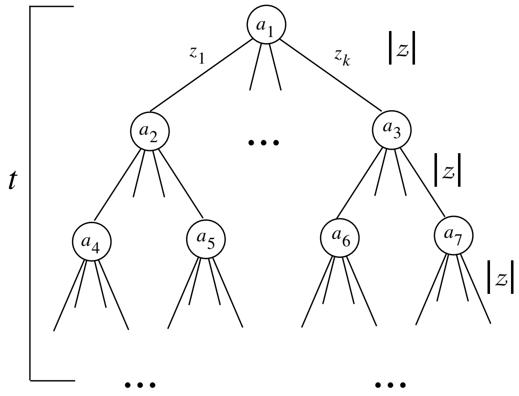

Figure 7.3: A t-step policy tree.

on the next level, which determines the next action. This is a complete recipe for t steps of conditional behavior.

Now, what is the expected discounted value to be gained from executing a policy tree p? It depends on the true state of the environment when the agent starts. In the simplest case, p is a 1-step policy tree (a single action). The value of executing that action in state s is $V_p(s) = R(s, \mathbf{act}(p))$, where $\mathbf{act}(p)$ is the action specified in the top node of policy tree p. More generally, if p is a t-step policy tree, then

$$\begin{aligned} V_p(s) &= R(s, \mathbf{act}(p)) + \beta \quad \text{Expected value of the future} \\ &= R(s, \mathbf{act}(p)) + \beta \sum_{s' \in \mathcal{S}} \Pr(s'|s, \mathbf{act}(p)) \sum_{z \in \mathcal{Z}} \Pr(z|s', \mathbf{act}(p)) V_{\mathbf{subtree}(p,z)}(s') \\ &= R(s, \mathbf{act}(p)) + \beta \sum_{s' \in \mathcal{S}} T(s, \mathbf{act}(p), s') \sum_{z \in \mathcal{Z}} O(s', \mathbf{act}(p), z) V_{\mathbf{subtree}(p,z)}(s') \\ &= R(s, \mathbf{act}(p)) + \beta \sum_{z \in \mathcal{Z}} \mathbf{stval}(\mathbf{act}(p), z, \mathbf{subtree}(p, z))[s], \end{aligned} \tag{7.1}$$

where $\mathbf{subtree}(p, z)$ is the $(t-1)$-step policy subtree associated with observation z at the top level of a t-step policy tree p, and $\mathbf{stval}(a, z, p')[s]$ is the probability-weighted value contributed by the subtree p' in the context of a policy tree with action a at the root when p' is the subtree for observation z:

$$\mathbf{stval}(a, z, p')[s] = \sum_{s' \in \mathcal{S}} T(s, a, s') O(s', a, z) V_{p'}(s').$$

Although this quantity has minimal intuitive appeal, it plays a crucial role in several of the algorithms.

Because we will never know the exact state of the environment, we must be able to determine the value of executing a policy tree p, from some information state x. This

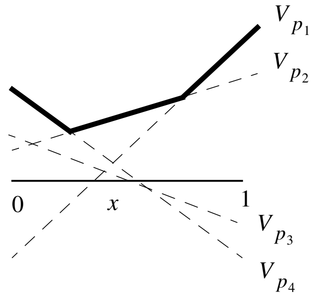

Figure 7.4: The optimal t-step value function is the upper surface of all the value functions associated with t-step policy trees.

is just an expectation over states of executing p in each state, which can be computed as a dot product:

$$V_p(x) = \sum_{s \in \mathcal{S}} x[s] V_p(s).$$

Now we have a function that represents the value of executing policy tree p in every possible information state. To construct an optimal t-step policy, however, it will generally be necessary to execute different policy trees depending on the initial information state. Let $\mathbb{P}_t$ be the finite set of all t-step policy trees. Then

$$V_t(x) = \max_{p \in \mathbb{P}_t} \sum_s x[s] V_p(s).$$

That is, the optimal t-step value of starting in information state x is the value of executing the policy tree that is best in x.

This definition of the value function leads to some important geometric insights into its form. Each policy tree p induces a value function that is linear in x, and the optimal t-step value function V_t is the upper surface of those functions. So, V_t is piecewise-linear and convex, as illustrated in Figure 7.4. Consider a POMDP with only two states. Each information state for the POMDP can be written as a vector of two non-negative numbers, $\langle x[s_1], x[s_2] \rangle$, that sum to 1: it has only one degree of freedom. The value function associated with a policy tree p_1, V_{p_1}, is a linear function of $x[s_1]$ and is shown in the figure as a line. The value functions of other policy trees are similarly represented. Finally, V_t is the maximum of all the V_{p_i}'s at each information state, giving us the upper surface, which appears in the figure as a bold line.

When there are three states in the environment, an information state is determined by two values. The space of information states can be visualized as a triangle in two-space with vertices $(0, 0)$, $(1, 0)$, and $(0, 1)$. The value function associated with a single

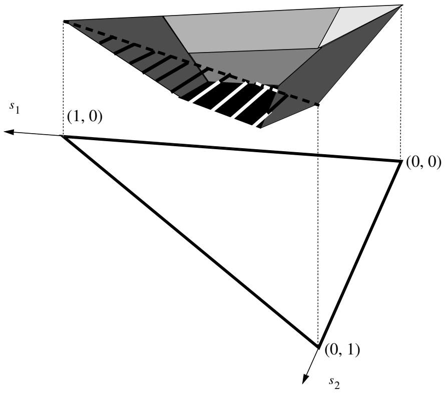

Figure 7.5: A value function in three dimensions.

policy tree is a plane in three-space, and the optimal value function is typically a bowl shape that is composed of planar facets; an example is shown in Figure 7.5. This general pattern holds in higher dimensions, but becomes difficult to contemplate and even harder to draw!

The convexity of the optimal value function makes intuitive sense when we think about the value of different information states. Information states that are at the corners of the space $\mathcal{X}$ of information states correspond to situations in which the agent is certain of the true underlying state. These information states have relatively high values (unless they correspond to states that are extremely undesirable), whereas information states closer to the "middle" correspond to high uncertainty situations in which it is more difficult for the agent to select actions appropriately to gain long-term reward.

Table 7.1 shows how policy trees can be used as a basis for a value-iteration algorithm. For each t, the policy-tree method enumerates the set $\mathbb{P}_t$ of all t-step policy trees, and then calls the function `BellmanErrMag` to determine whether the value functions represented by $\mathbb{P}_t$ and $\mathbb{P}_{t-1}$ are close together. A linear-programming algorithm for `BellmanErrMag` is given in Section G.1.

The policy-tree method is, of course, hopelessly computationally intractable. Each t-step policy tree contains $(|\mathcal{Z}|^t - 1)/(|\mathcal{Z}| - 1)$ nodes (the branching factor is $|\mathcal{Z}|$, the number of possible observations). Each node can be labeled with one of $|\mathcal{A}|$ possible actions, so the total number of t-step policy trees is

$$|\mathcal{A}|^{\frac{|\mathcal{Z}|^t - 1}{|\mathcal{Z}| - 1}},$$

```
PolicyTreeMethod(⟨𝒮, 𝒜, T, R, 𝒵, O, β⟩, ε) := {
    t := 0
    loop
        t := t + 1
    until BellmanErrMag(ℙ_t, ℙ_{t−1}) < ε
    return ℙ_t
}
```

Table 7.1: Value iteration using the policy-tree method.

which grows astronomically in t.

It is not known whether any algorithm for computing a set of policy trees to represent the t-step value function has a better worst-case run time. This is because the number of policy trees needed to represent the t-step value function might actually be doubly exponential in t in the worst case. For solving a finite-horizon POMDP with a given initial belief state, the PSPACE finite-horizon algorithm of Papadimitriou and Tsitsiklis [116], mentioned in Section 6.3.3, can be made to run in singly exponential time in t, which is better in the worst case. In the next few sections, I present algorithms that run faster for POMDPs that possess simple value functions.

7.3.2 A Note on Implementation

Several of the algorithms in this section make use of sets of policy trees as a primitive data structure. Policy trees can be represented by a tree-like data structure; however, for efficiency of space and computation speed, other data structures might be preferred. The policy-tree data structure needs to support the operators defined in Table 7.2.

All the necessary primitive operations on policy trees can be implemented on a data structure that consists of a vector of values, an action, and a pointer for each observation to a vector of values or a policy tree.

7.3.3 Useful Policy Trees

In general, the set $\mathbb{P}_t$ of t-step policy trees contains many policy trees whose value functions are totally dominated by or tied with value functions associated with other policy trees. Figure 7.6 shows a situation in which the value function associated with policy tree p_d is completely dominated by (everywhere less than or equal to) the value

$\mathbf{tree}(a, \tau)$	create a new policy tree with action a at root and subtree for observation z equal to $\tau(z)$
V_p	return a vector representing the value function for policy tree p with one component per state
$\mathbf{act}(p)$	return the action at the root of policy tree p
$\mathbf{subtree}(p, z)$	return the subtree of policy tree p associated with observation z; a subtree can be a policy tree, or more simply, the value function of the subtree
$\mathbf{stval}(a, z, p)$	return a vector representing the probability-weighted value of following policy tree p as the observation z subtree of a policy tree with a at the root
$\succ, \succ_{a,z}$	compare policy trees lexicographically according to their value functions

Table 7.2: A list of operations needed for policy-tree-based algorithms.

function for policy tree p_b. The situation with the value function for policy tree p_c is somewhat more complicated; although it is not completely dominated by any single value function, it *is* completely dominated by p_a and p_b taken together.

Given a set G of policy trees representing a piecewise-linear convex value function, it is possible to define a minimal subset Γ that represents the same function that G represents. I call the elements of this set the *useful* policy trees; it is unique up to substitutions of policy trees with exactly the same value function. In the following discussion, I assume that no two policy trees in G have the same value function. In practice, it is easy to examine a set of policy trees and to throw out all but one policy tree for each value function represented. This does not change the piecewise-linear convex function represented and guarantees that no two policy trees yield identical value functions (an implementation appears in Table 7.3).

Using the definition of useful policy trees directly, it appears that to determine whether a policy tree is useful, we must perform a combinatorial search for the set Γ. The following lemma shows that usefulness is a property of the individual vectors in G.

Lemma 7.1 *Let G be a set of policy trees. A policy tree $p \in G$ is useful if and only if there is some information state x such that $V_p(x)$ is strictly greater than $V_{\tilde{p}}(x)$ for all other policy trees $\tilde{p} \in G$.*

Proof: A proof is given in Section G.2. $\square$

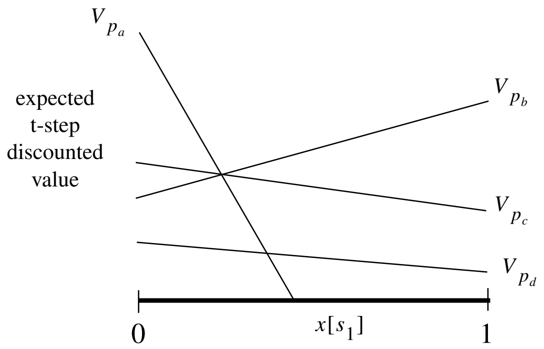

Figure 7.6: Some policy trees may be totally dominated by others and can be ignored.

`uniq`$(G) := \{$
 $G' := \emptyset$
 while $(G \neq \emptyset)$ $\{$
  $p :=$ any element in G
  $G' := G' \cup \{p\}$
  $G := G - \{p\}$
  foreach $(\tilde{p} \in G)$
   if $(V_p = V_{\tilde{p}})$ $G := G - \{\tilde{p}\}$
 $\}$
 return G'
$\}$

Table 7.3: Subroutine for removing policy trees from G so that any pair of policy trees remaining have different value functions.

```
Filter(G) := {
    G := uniq(G)
    Γ := ∅
    foreach p ∈ G
        if (dominate(p, G) ≠ false) Γ := Γ ∪ {p}
    return Γ
}
```

Table 7.4: Subroutine for returning the useful policy trees in G.

```
dominate(p, G) := {
    if (G = ∅) then return any element in 𝒳
    Solve the following linear program:
        maximize: d
        s.t.: Σ_s x[s]V_p(s) ≥ Σ_s x[s]V_p̃(s) + d, for all p̃ ∈ G − {p}
        and: Σ_s x[s] = 1
        and: x[s] ≥ 0, for all s ∈ 𝒮
        variables: d, x[s] for all s ∈ 𝒮
    if (d > 0) then return x
    else return false
}
```

Table 7.5: Subroutine for finding an information state at which policy tree p dominates all other policy trees in G.

Lemma 7.1 gives us a way of computing the set Γ of useful policy trees, the minimal set of policy trees needed to represent the value function for G. We need only loop over the policy trees in G, testing whether there is an x where each policy tree dominates the others (an implementation appears in Table 7.4). The domination condition itself can be checked using linear programming (an implementation appears in Table 7.5). The linear program can take many different forms; as presented here, a variable d is used to represent the amount by which a policy tree p dominates all the policy trees in G at information state x. Maximizing d results in the identification of the x at which p dominates the policy trees in G the most. If the maximum amount is negative or zero, p does not dominate all the policy trees in G anywhere and is therefore not a member of Γ.

```
EnumerationMethod(⟨S, A, T, R, Z, O, β⟩, ε) := {
    t := 0
    Γ_0 := ∅
    loop
        t := t + 1
        G_t := {tree(a, τ)|a ∈ A, τ ∈ T(Z → Γ_{t-1})}
        Γ_t := Filter(G_t)
    until BellmanErrMag(Γ_t, Γ_{t-1}) < ε
    return Γ_t
}
```

Table 7.6: Value iteration in information-state MDPs using the enumeration method.

The ability to compute the set of useful policy trees serves as a basis for a more efficient version of the value-iteration algorithm in Table 7.6, generally attributed to Monahan [109].

Some new notation is introduced in Table 7.6. First, $\mathcal{T}(\mathcal{Y} \rightarrow \Phi)$ represents the set of all mappings from a finite set $\mathcal{Y}$ to a finite set Φ. For $\tau \in \mathcal{T}(\mathcal{Y} \rightarrow \Phi)$, $\tau(y) \in \Phi$ for all $y \in \mathcal{Y}$. There are $|\Phi|^{|\mathcal{Y}|}$ elements in the set $\mathcal{T}(\mathcal{Y} \rightarrow \Phi)$ and they can be enumerated easily.

Second, $\mathbf{tree}(a, \tau)$ is the t-step policy tree with action a at its root, and a policy subtree for each observation $z \in \mathcal{Z}$ equal to $\tau(z)$, where $\tau \in \mathcal{T}(\mathcal{Z} \rightarrow \mathbb{P}_{t-1})$. As a demonstration of this new notation, here is a recursive definition for the set of t-step policy trees:

$$\mathbb{P}_t = \{\mathbf{tree}(a, \tau) | a \in \mathcal{A}, \tau \in \mathcal{T}(\mathcal{Z} \rightarrow \mathbb{P}_{t-1})\}.$$

7.3.4 The Enumeration Method

The enumeration method is used by a family of algorithms that exploit the following idea: Γ_{t-1}, the set of useful policy trees for the $(t-1)$-step value function, can be used to construct a superset G_t of the useful t-step policy trees. In constructing the policy trees in G_t, the choice of subtree is restricted to those $(t-1)$-step policy trees that were useful. This is justified by the fact that, for any information state and any choice of policy subtree, there is always a useful subtree that is at least as good at that state; there is never any reason to include a non-useful policy subtree.

The time complexity of a single iteration of this algorithm can be divided into two parts: generating G_t and filtering G_t. There are $|\mathcal{A}||\Gamma_{t-1}|^{|\mathcal{Z}|}$ elements in G_t because there are $|\mathcal{A}|$ different ways to choose the action, and $|\Gamma_{t-1}|^{|\mathcal{Z}|}$ different mappings from $\mathcal{Z}$ to Γ_{t-1} corresponding to the different combinations of subtrees. The value functions for the policy trees in G_t can be computed efficiently from those of the subtrees. Thus, generating G_t is exponential in the number of observations.

Filtering also takes exponential time, but even worse, it involves solving an exponential number of exponential-size linear programs. Although this algorithm may represent a large computational savings over the policy-tree method, it still does more work than may be necessary. The next section shows how the linear programs used to implement the filtering stage can be made significantly smaller.

7.3.5 Lark's Filtering Algorithm

The filtering algorithm in Table 7.4 uses the `dominate` subroutine to decide whether a policy tree p dominates all others in a set G. When there is no information state x at which p has a larger value than all policy trees in G, `dominate` returns `false`. However, when there is such an x, `dominate` returns it, but `Filter` ignores its actual value.

A filtering algorithm attributed to Lark [176] shows how the useful policy trees can be identified one by one by making use of the identity of the information state x at which one policy tree dominates the others. As a result, the size of the linear programs used to test domination can be bounded by the size of the set of useful policy trees in G, instead of the size of G itself.

Lark's filtering algorithm uses the following insight. Let U be a set of policy trees that have been determined to be useful. The set U does not equal the complete set Γ of useful policy trees if and only if some policy tree $p \in G$ dominates the policy trees in U.

The algorithm maintains a set U of policy trees that have been determined to be useful, and a set `unchecked` of policy trees that have not yet been determined to *not* be useful. An iteration of the algorithm proceeds by choosing a policy tree p from `unchecked` and checking whether there is an x at which it dominates all the useful policy trees in U. If no such x exists, then p is not useful and is removed from `unchecked`.

If there is an x at which p dominates the policy trees in U, then there is at least one useful policy tree still missing from U. The missing policy tree is not necessarily p: we know that p dominates the policy trees in U at x, not that it dominates all the policy trees in G at x. The following lemma provides one way we can use x to identify a policy tree that is guaranteed to be useful.

Lemma 7.2 *Given a set of policy trees G and an information state x, let p^* be the policy tree in G that has the largest value at x where ties are broken in favor of the policy tree with the lexicographically greater value vector. Then p^* is useful with respect to G.*

Proof: A proof appears in Section G.2. □

We say that one vector is *lexicographically greater than* another vector if, given some predetermined ordering over the states in $\mathcal{S}$, the first vector has a larger first component, or the two vectors are tied on their first i components and the first vector is larger in component $i+1$. We can use this to define an ordering relation over policy trees: $p_1 \succ p_2$ if the vector of values V_{p_1} is lexicographically greater than the vector of values V_{p_2}.

The subroutine `best` in Table 7.7 chooses a policy tree from a set P that has maximum value at an information state x and is guaranteed to be useful. The subroutine in Table 7.8 makes use of `best` to identify useful policy trees from a set. As described above, each call to `dominate` is given only the set of known useful policy trees. As a result, no linear program larger than the full set of useful policy trees is constructed. Using `FilterLark` in place of `Filter` in the value-iteration algorithm in Table 7.6 results in a much faster algorithm [95].

7.3.6 The Witness Algorithm

The POMDP value-iteration algorithms discussed in the previous sections suffer from the problem that the set G_t of possibly useful policy trees is constructed at each step. Since the size of G_t is exponential in the number of observations, these algorithms are terribly inefficient for solving POMDPs with more than a small number of observations ($|\mathcal{Z}| = 6$ appears to be a practical upper limit [95]).

If we hope to solve larger problems, we need to avoid generating G_t. The witness algorithm works by building up a set of useful policy trees, one by one, analogous to the way Lark's filtering algorithm operates, except without making use of an explicit

```
best(x, P) := {
    maxtree := any element in P
    maxval := Σ_s x[s] V_maxtree(s)
    foreach p ∈ P − {maxtree} {
        val := Σ_s x[s] V_p(s)
        if ((val > maxval) or ((val = maxval) and (p ≻ maxtree))) then {
            maxtree := p
            maxval := val
        }
    }
    return maxtree
}
```

Table 7.7: Subroutine for finding a useful policy tree at x, given a set of policy trees P.

```
FilterLark(G) := {
    U := ∅
    unchecked := G
    while (unchecked ≠ ∅) {
        p := any element in unchecked
        x := dominate(p, U)
        if (x = false) then unchecked := unchecked − {p}
        else {
            p* := best(x, unchecked)
            U := U ∪ {p*}
            unchecked := unchecked − {p*}
        }
    }
    return U
}
```

Table 7.8: Lark's method for computing the useful policy trees in G.

```
WitnessOuter(𝓜 = ⟨𝓢, 𝓐, T, R, 𝓩, O, β⟩, ε) := {
    Γ_0 := ∅
    t := 0
    loop
        t := t + 1
        foreach a ∈ 𝓐
            Γ_t^a := WitnessInner(a, Γ_{t-1}, 𝓜)
        Γ_t := FilterLark(⋃_a Γ_t^a)
    until BellmanErrMag(Γ_t, Γ_{t-1}) < ε
    return Γ_t
}
```

Table 7.9: Value iteration in information-state MDPs using the witness algorithm.

representation of G_t. When Lark's filtering algorithm is used in the context of value iteration, it uses the set G_t in two ways. First, it uses G_t as a source of policy trees that might reveal that U is incomplete; once all the policy trees in G_t are considered, Lark's algorithm terminates. Second, G_t is searched to identify a useful policy tree given an information state x. The witness algorithm avoids both of these uses of G_t.

The main difference between the high-level structure of the witness algorithm and that of the algorithms mentioned earlier is that the witness algorithm first finds a representation for the t-step Q functions. As a result, the outer loop of the witness algorithm (Table 7.9) closely resembles the value-iteration algorithm for MDPs in Table 2.2, with the set Γ_t^a playing the role of the Q function $Q_t(\cdot, a)$.

By arguments parallel to those in Section 7.3.1, the t-step Q function for action a is piecewise linear and convex, and can be represented by a minimal set of policy trees, Γ_t^a. Because the value of an information state is the maximum Q value for that state, $V(x) = \max_a Q(x, a)$, it must be the case that every policy tree p in the set Γ_t of useful policy trees for the t-step value function is in $\Gamma_t^{\mathbf{act}(p)}$. Therefore, we can compute Γ_t given the Γ_t^a sets by finding the useful vectors in $\bigcup_a \Gamma_t^a$, which might be a good deal larger than Γ.

Any of the algorithms I mentioned earlier can be used to construct Γ_t^a; however, all need to construct the exponential-size set of possibly useful policy trees. To build up to Γ_t^a without enumerating an exponential-size set, we need to answer two questions: "How do we find useful policy trees without enumerating all policy trees?" and "How

```
UsefulPolicyTreeFromState(x, a, Γ_{t-1}, M = ⟨S, A, T, R, Z, O, β⟩) := {
    foreach z ∈ Z
        τ(z) := bestSubtree(x, a, z, Γ_{t-1}, M)
    return tree(a, τ)
}

bestSubtree(x, a, z, P, ⟨S, A, T, R, Z, O, β⟩) := {
    maxtree := any element in P
    maxval := Σ_s x[s]stval(a, z, maxtree)[s]
    foreach p ∈ P {
        val := Σ_s x[s]stval(a, z, p)[s]
        if ((val > maxval) or ((val = maxval) and (p ≻_{a,z} maxtree))) then {
            maxtree := p
            maxval := val
        }
    }
    return maxtree
}
```

Table 7.10: Computing a useful policy tree at x, given action a.

will we know when we are done?"

The first question was answered in the context of Smallwood and Sondik's [148] POMDP algorithm. The subroutine UsefulPolicyTreeFromState in Table 7.10 shows how to construct a useful (with respect to Γ_t^a) t-step policy tree for action a that is useful at information state x, given the set Γ_{t-1}. It works much like the implementation of best in that it identifies the policy tree with maximum value at x, breaking ties using lexicographic ordering. Instead of considering each candidate policy tree separately, it constructs one directly.

To see how UsefulPolicyTreeFromState works, first notice that we can build a policy tree with maximum value at x by maximizing the subtree values. If $\mathbb{P}_t^a$ is the set of t-step policy trees with action a at the root, then the value of the best policy tree at x is

$$\begin{aligned}
\max_{p \in \mathbb{P}_t^a} V_p(x) &= \max_{p \in \mathbb{P}_t^a} \left(\sum_s x[s] \left(R(s,a) + \beta \sum_z \mathbf{stval}(a, z, \mathbf{subtree}(p, z))[s] \right) \right) \\
&= \sum_s x[s] \left(R(s,a) + \beta \sum_z \max_{p_z \in \mathbb{P}_{t-1}} \mathbf{stval}(a, z, p_z)[s] \right) \\
&= \sum_s x[s] R(s,a) + \beta \sum_z \max_{p_z \in \Gamma_{t-1}} \sum_s x[s] \mathbf{stval}(a, z, p_z)[s]. \qquad (7.2)
\end{aligned}$$

This is justified by the formula for V_p in Equation 7.1, and the fact that Γ_{t-1} is the set of useful $(t-1)$-step policy trees. Equation 7.2 essentially says that we can choose the subtree for each observation separately. The code in Table 7.10 implements this idea, choosing the best subtree for each observation using `bestSubtree`. The `bestSubtree` subroutine works much like `best`, choosing a policy tree with maximum (subtree) value with respect to x, breaking ties lexicographically (the relation $p_1 \succ_{a,z} p_2$ is true if the vector $\mathbf{stval}(a, z, p_1)$ is lexicographically greater than $\mathbf{stval}(a, z, p_2)$). Close examination of Equation 7.2 reveals that breaking ties for each observation subtree in favor of the lexicographic maximum yields the lexicographically largest policy tree when the subtrees are combined. Therefore, by Lemma 7.2, `UsefulPolicyTreeFromState` returns a useful policy tree, even in the case of ties, without enumerating an exponential-size set.

To answer the question "How will we know when we are done?" we need some additional terminology. Policy trees p_1 and p_2 are *neighbors* if $\mathbf{act}(p_1) = \mathbf{act}(p_2)$, and $\mathbf{subtree}(p_1, z) = \mathbf{subtree}(p_2, z)$ for all but one $z \in \mathcal{Z}$. Each t-step policy tree has $|\mathcal{Z}|(|\Gamma_{t-1}| - 1)$ neighbors, which can be enumerated easily. The following lemma forms the basis of a termination test.

Lemma 7.3 *Let U be a set of policy trees that have been determined to be useful with respect to action a. The set U does not equal the complete set Γ^a of useful policy trees if and only if some policy tree p, in the set of neighbors of policy trees in U, dominates the policy trees in U.*

Proof: A proof appears in Section G.5. □

Lemma 7.3 is quite powerful because it lets us determine whether a subset U of useful policy trees is complete by examining only the relatively small set of neighboring policy trees.

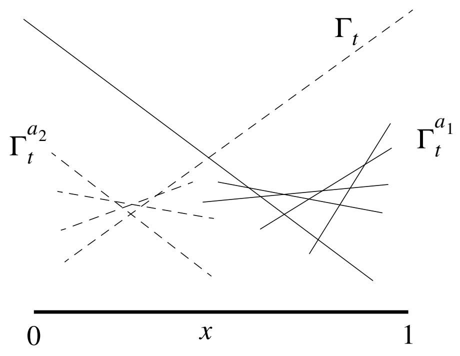

Figure 7.7: Q functions can be arbitrarily more complex than their corresponding value functions.

The code in Table 7.11 builds up the set of useful policy trees. The set `unchecked` is an agenda, initialized with a single arbitrary policy tree. Each iteration takes a policy tree p off the agenda and determines whether there is an information state x that can "witness" the fact that p dominates the policy trees in U. If such an x is discovered, its associated policy tree is added to U and all neighbors of the policy tree are added to the agenda. If p does not dominate the policy trees in U, then p is removed from the agenda. When the agenda is empty, the algorithm terminates.

Because it only ever constructs the neighbors of the useful policy trees (and not all possibly useful policy trees), the witness algorithm runs very efficiently over a wide range of POMDPs. Like the enumeration algorithms, however, the witness algorithm may do more work than is necessary. In particular, the witness algorithm spends a great deal of time finding the exact set of policy trees needed to represent the Q functions, when, in fact, many of these policy trees may not be useful when they are pooled to form the optimal value function; Figure 7.7 is an example value function in which the number of policy trees in the optimal Q function is much larger than the number of policy trees in the optimal value function. It would be desirable to identify a lemma analogous to Lemma 7.3 that pertains to value functions instead of Q functions. No such lemma is known, and there are complexity-theoretic reasons to believe that it may not exist (see Section 7.5).

7.3.7 Other Methods

Several other algorithms have been proposed to perform value-iteration updates in information-state MDPs. Sondik [149] proposed the first such algorithm. Although

```
WitnessInner(a, Γ_{t-1}, M = ⟨S, A, T, R, Z, O, β⟩) := {
    U := ∅
    unchecked := {any element in ℙ_t}
    while (unchecked ≠ ∅) {
        p := any element in unchecked
        x := dominate(p, U)
        if (x = false) then unchecked := unchecked − {p}
        else {
            p* := UsefulPolicyTreeFromState(x, a, Γ_{t-1}, M)
            U := U ∪ {p*}
            unchecked := unchecked ∪ neighbors(p*, M)
        }
    }
    return U
}

neighbors(p, Γ_{t-1}, ⟨S, A, T, R, Z, O, β⟩) := {
    U := ∅
    foreach z ∈ Z τ(z) := subtree(p, z)
    foreach z ∈ Z {
        foreach p' ∈ Γ_{t-1} − {subtree(p, z)} {
            τ(z) := p'
            U := U ∪ tree(act(p), τ)
        }
        τ(z) := subtree(p, z)
    }
    return U
}
```

Table 7.11: Computing the set of useful t-step policy trees for action a, via the witness algorithm.

the algorithm is complicated and, in principle, avoids enumerating the set of possibly useful policy trees at each iteration, it appears to run more slowly than the simpler enumeration methods in practice [33].

Cheng [33] developed a collection of algorithms for solving POMDPs. His relaxed region and linear support algorithms work by building up a set U of useful policy trees using specialized algorithms for enumerating the *extreme points* in the sets of information states over which each $p \in U$ dominates. The algorithms run very quickly when $|\mathcal{S}|$ is small, but scale poorly because the number of extreme points can grow exponentially with the size of the state space.

White and Scherer [177] propose an alternative approach in which the reward function is changed so that all of the algorithms discussed in this chapter will tend to run more efficiently. This technique has not yet been combined with the witness algorithm, and may provide some improvement.

7.4 Algorithmic Analysis

An information-state MDP is a generalized MDP with an infinite state space and maximum expected value optimality criterion. Theorem 3.2 bounds the number of iterations needed for value iteration to identify a value function with an ϵ-optimal policy.

In this section, I analyze the time complexity of a single step of value iteration for several of the algorithms described in Section 7.3. I refer to the problem of computing the set Γ_t of useful policy trees from a set Γ_{t-1} of vectors as the *one-stage* POMDP *problem.* The sets Γ_t and Γ_{t-1} represent the t-step and $(t-1)$-step value function, respectively. The size of a one-stage POMDP problem is equal to $|\mathcal{S}| + |\mathcal{Z}| + |\mathcal{A}|$ for the state, observation and actions sets; plus the size of $|\mathcal{S}|^2|\mathcal{A}| + |\mathcal{S}||\mathcal{A}| + |\mathcal{S}||\mathcal{A}||\mathcal{Z}| + 1$ rational numbers for the transition function, observation function, reward function, and discount factor; plus the size of $|\Gamma_{t-1}||\mathcal{S}|$ rational numbers for the $(t-1)$-step value function.

Although the algorithms described earlier use policy trees to represent the $(t-1)$-step value function, it is not difficult to adapt them to work directly with sets of vectors. I use this model here because it makes it easier to construct examples with particular properties. I abuse notation and write Γ_{t-1} for the set of vectors representing V_{t-1}, instead of the set of policy trees.

7.4.1 Enumeration Algorithms

Several algorithms for finding Γ_t work by enumerating the set G_t of possibly useful policy trees, and then identifying which of these policy trees is useful: Monahan's algorithm [109] was the first and later Eagle [51] and Lark [176] provided improvements. However, all these algorithms, regardless of their details, build G_t, the size of which is $|\mathcal{A}||\Gamma_{t-1}|^{|\mathcal{Z}|}$. Thus, even if a policy tree could be identified as useful in constant time, the run times of these algorithms are at least exponential in $|\mathcal{Z}|$, making them of little use for solving POMDPs with anything but the smallest observation sets.

7.4.2 The One-pass Algorithm

Sondik's one-pass algorithm [149, 148] was the first exact algorithm for solving finite-horizon POMDPs. At a high level, the algorithm works by taking a useful policy tree p and constructing a set of linear constraints over the set of information states that guarantee that p will be the optimal policy tree throughout the constrained region. There is one constraint for each policy tree p_a obtained by substituting action a for the root of p, plus one for each neighbor of the p_a trees. By identifying the optimal policy tree in each region adjacent to the constrained region, a systematic search for optimal policy trees can be carried out.

Because of the complicated nature of the algorithm, and its poor performance in empirical evaluations [33], I will not present a detailed analysis of the one-pass algorithm. However, it is possible to construct POMDPs in which it is necessary to create all possible constraint sets; as a result, the worst-case run time of the one-pass algorithm is at least $(|\mathcal{A}||\Gamma_{t-1}|)^{|\mathcal{Z}|}$ iterations, which can be considerably worse than the worst-case bound for enumeration algorithms.

7.4.3 Extreme-point Algorithms

Cheng's linear support and relaxed region algorithms [33] make use of special-purpose routines that enumerate the vertices of each linear region of the value function.

Bounding the number of vertices in a polyhedron is a well-studied problem [80] and it is known that there can be an exponential number. In fact, there is a family of one-stage POMDP problems such that, for every n, $|\mathcal{S}| = n + 1$, $|\mathcal{A}| = 2n + 1$, $|\mathcal{Z}| = 1$, $|\Gamma_{t-1}| = 1$, $|\Gamma_t| \leq 2n + 1$, and yet the number of vertices in one of the regions is 2^n. The construction is given in Section G.3. Since visiting each vertex is just one of the operations the extreme-point algorithms perform, we can expect the worst-case run time to grow at least exponentially in the size of the one-stage POMDP problem.

7.4.4 The Witness Algorithm

This section contains a run-time analysis of the witness algorithm on one-stage POMDP problems, in terms of the size of the problem and $\sum_a |\Gamma_t^a|$, the size of the sets of useful policy trees for each action. The run time is polynomial in these quantities, although it is not difficult to construct examples in which $\sum_a |\Gamma_t^a|$ is exponential in the size of the one-stage POMDP problem.

At the highest level, the witness algorithm computes Γ_t^a for each $a \in \mathcal{A}$, and then selects Γ_t from the union of the Γ_t^a sets. In computing Γ_t^a, the total number of policy trees added to `unchecked` is equal to the number of neighbors of the policy trees used to construct the vectors in Γ_t^a plus the arbitrarily chosen starting policy tree, specifically, $1+|\mathcal{Z}|(|\Gamma_{t-1}|-1)|\Gamma_t^a|$. Each pass through the "while" loop in the inner loop (Table 7.11) either consumes an element from `unchecked` ($1 + |\mathcal{Z}|(|\Gamma_{t-1}| - 1)|\Gamma_t^a|$ times) or adds a vector to U ($|\Gamma_t^a|$ times). Thus, the total number of iterations in `WitnessInner` is

$$1 + |\mathcal{Z}|(|\Gamma_{t-1}| - 1)|\Gamma_t^a| + |\Gamma_t^a|.$$

The statements in the loop in `WitnessInner` can all be implemented to run in polynomial time; this includes `dominate`, since polynomial-time algorithms for linear programming with polynomial-precision rational numbers exist [140]. The total run time of `WitnessInner` for each a is therefore bounded by a polynomial in the size of the one-stage POMDP problem and $|\Gamma_t^a|$.

The `WitnessOuter` routine calls `WitnessInner` for each $a \in \mathcal{A}$ and then calls `FilterLark`, which creates one linear program for each policy tree found. For one-stage POMDP problems in which $\sum_a |\Gamma_t^a|$ is polynomially bounded, this implies that the total run time is polynomial. The algorithm takes no more than exponential time in the worst case because $\sum_a |\Gamma_t^a| \leq |G_t| = |\mathcal{A}||\Gamma_t|^{|\mathcal{Z}|}$.

7.5 Complexity Results

In this section, I present some results pertaining to the computational complexity of the one-stage POMDP problem described in the previous section.

It is not difficult to show that *no* algorithm can compute Γ_t from Γ_{t-1} in polynomial time for general POMDPs, simply because Γ_t can be exponentially large with respect to the size of the one-stage POMDP problem. An example POMDP illustrating this phenomenon is presented in Section G.3.

Any algorithm for computing Γ_t in polynomial time must only apply to a subclass of POMDPs. We call a family of one-stage POMDP problems *polynomially output bounded* if $|\Gamma_t|$ can be bounded by a polynomial in the size of the POMDP and Γ_{t-1}.

No existing algorithm has been shown to run in polynomial time on polynomially output-bounded one-stage POMDP problems, and the next theorem suggests that there may be a good reason for this.

Theorem 7.1 *The best algorithm for solving polynomially output-bounded one-stage* POMDP *problems runs in polynomial time if and only if RP=NP.*

Proof: The theorem is proved in Section G.4. □

The importance of Theorem 7.1 is that it links the problem of exactly solving one-stage POMDPs with the complexity-theoretic question of whether RP=NP.

These results imply that further restrictions on the class of one-stage POMDP problems are needed before a polynomial-time algorithm will be found. A family of one-stage POMDP problems is *polynomially action-output bounded* if $\sum_{a \in A} |\Gamma_t^a|$ is bounded by a polynomial in the size of the one-stage POMDP problem. As before, Γ_t^a is the minimum set of policy trees needed to represent the t-step Q function for action a.

The quantity $\sum_{a \in A} |\Gamma_t^a|$ is an upper bound on $|\Gamma_t|$, though the bound may be arbitrarily loose. By focusing on polynomially action-output-bounded POMDPs, we can solve for Γ_t in polynomial time as long as we can find Γ_t^a in polynomial time for each $a \in A$.

The performance of the algorithms described in this chapter on this restricted class of POMDPs is summarized in the following theorem.

Theorem 7.2 *Of the existing algorithms that can be used to solve polynomially action-output bounded* POMDP*s, only the witness algorithm runs in polynomial time.*

Proof: The theorem follows from the run-time analyses in Section 7.4. □

7.6 Reinforcement Learning in Information-state MDPs

I now briefly describe several approaches to learning a policy for an information-state MDP from experience. This differs from learning a policy for a POMDP from experience (see Section 6.6) in that here the experience tuples have the form: $\langle x, a, x', r \rangle$; the information states are provided instead of observations. This type of experience tuple would arise in situations where the rewards and transition probabilities are known in advance or when an accurate model of the environment has been learned on line.

This is an interesting application of reinforcement learning, because the model is known in advance and yet there are still sound reasons for trying to learn the optimal Q function from experience; for example, reinforcement learning could possibly find a useful approximation over the important parts of the state space more quickly than the analytical methods described earlier. Still, there are major challenges to applying any of the algorithms discussed earlier to learning in an information-state MDP, most particularly the fact that the optimal Q function cannot be represented by a table of values.

In this section, I sketch several methods for finding linear or piecewise-linear convex approximations to the optimal Q functions for POMDPs. In each, a simple, parameterized function representation is used to approximate the optimal Q function for each action. In some cases, the parameterized function is linear, that is, a set of coefficients, one for each state; it assigns values to information states by taking the dot product of the information state and the coefficients. The approximate value function for an information state is the maximum value assigned to that state by any of the Q functions; this means that if the Q functions are approximated by linear or piecewise-linear convex functions, the approximate value function will be piecewise linear and convex.

7.6.1 Replicated Q-learning

As described in Section 6.6, Chrisman [34] and McCallum [104] explored the problem of learning a POMDP model in a reinforcement-learning setting. At the same time that their algorithms attempt to learn the transition and observation probabilities, an extension of Q-learning [173] was used to approximate Q functions for the learned POMDP model. Although it was not the emphasis of their work, their *replicated Q-learning* rule is of independent interest.

Replicated Q-learning generalizes Q-learning to apply to vector-valued states and uses a single vector, q_a, to approximate the Q function for each action a: $Q(x,a) = \sum_s x[s] q_a[s]$.

The components of the vectors are updated using the rule

$$\Delta q_a[s] = \alpha\ x[s] \left(r + \beta \max_{a'} Q(x', a') - q_a[s] \right) .$$

The update rule is applied for every $s \in \mathcal{S}$ each time the agent makes a state transition; α is a learning rate, x an information state, a the action taken, r the reward received, and x' the resulting information state. This rule applies the Q-learning update rule to each component of q_a in proportion to the probability that the agent is currently occupying the state associated with that component.

This learning rule can be applied to the problem of solving information-state MDPs. If the observations of the POMDP are sufficient to ensure that the agent is always certain of its state (i.e., $x[s] = 1$ for some s at all times), this rule reduces exactly to standard Q-learning and existing convergence theorems apply (see Section 2.6.1).

The rule itself is an extremely natural extension of Q-learning to vector-valued state spaces. In fact, an elaboration of this rule was developed independently by Connell and Mahadevan [38] for solving a distributed-representation reinforcement-learning problem in robotics.

7.6.2 Linear Q-learning

Although replicated Q-learning is a generalization of Q-learning, it does not extend correctly to cases in which the agent is faced with significant uncertainty. Consider a POMDP in which the optimal Q function can be represented with a single linear function. Since replicated Q-learning independently adjusts each component of the approximate linear representation of the Q function to predict the moment-to-moment Q values, the learning rule tends to move all components of q_a toward the same value.

The components of q_a ought to be set to match the coefficients of the linear function that predicts the Q values. This suggests using the delta rule for neural networks [131], which, adapted to the information-state MDP framework, becomes:

$$\Delta q_a[s] = \alpha\ x[s] \left(r + \gamma \max_{a'} Q(x', a') - Q(x, a) \right) .$$

Like the replicated Q-learning rule, this rule reduces to ordinary Q-learning when the information state is deterministic.

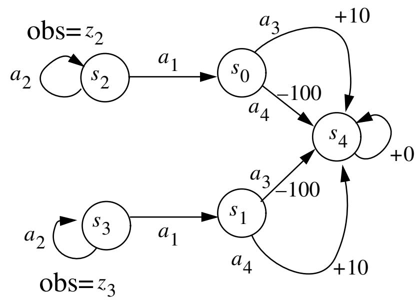

Figure 7.8: A POMDP that cannot be solved with a single linear function per action. Unmarked rewards are zero, unmarked observations are z_1, unmarked transitions are self transitions, and the initial information state has equal probability on states s_2 and s_3.

In neural network terminology, linear Q-learning views $\{x,\ r + \beta \max_{a'} Q(x', a')\}$ as a training instance for the function $Q(\cdot, a)$. Replicated Q-learning, in contrast, uses this example as a training instance for the component $q_a[s]$ for every s. The rules behave significantly differently when the components of q_a need to have widely different values to solve the problem at hand [94].

7.6.3 More Advanced Representations

Although replicated Q-learning and linear Q-learning seem to work quite well on small problems, the linear functions they use are not adequate in general. As mentioned earlier, piecewise-linear convex functions can approximate the optimal Q functions as closely as necessary. In contrast, the linear functions used by the learning algorithms can result in arbitrarily bad approximations.

As a concrete example of a POMDP that cannot be solved using simple linear Q functions, consider the POMDP illustrated in Figure 7.8. A policy is *linearly representable* if it can be represented as the greedy policy with respect to some linear representation of the Q functions. I will show that the optimal policy for the POMDP of Figure 7.8 is not linearly representable, and therefore that a more complex representation is needed to solve it.

As an aside, there is some connection between the notion of linear separability in classification tasks and linear representability of optimal POMDP policies. Indeed, the argument that the optimal policy for the POMDP of Figure 7.8 is not linearly representable fairly closely mimics the classic argument that "xor" is not linearly separable [66].

To show that the POMDP of Figure 7.8 is not linearly representable, I first describe the optimal policy for this environment and argue that it is unique, then show that any choice of a single vector to represent the Q values for action a_1 leads to a suboptimal decision for some information state on the path from the initial state to the goal state. The optimal policy can be represented using a single vector for actions a_2, a_3, and a_4 and two vectors for action a_1.

The unique optimal policy in this environment is to take action a_2 to determine whether the agent is in state s_2 (observation z_2) or state s_3 (observation z_3). If the agent is in state s_2, it needs to take action a_1, then a_3. If the agent is in state s_3, it needs to take action a_1, then a_4. The expected number of steps to goal for this policy is 3 and the value of the initial state is $10\beta^3$.

To see that no other policy does as well, note that actions a_3 and a_4 from the initial state are clearly suboptimal. If a_1 is selected as the initial action, the second action would have to be either a_3 or a_4. For either choice, half of the time this would lead the agent to a reward of -100, and the other half, the agent would receive 10. The average is then $-45 < 10\beta^3$; thus, the unique optimal policy is the one stated above.

Given that we know how to behave optimally for this POMDP, we now need to show that no single-vector-per-action representation can capture the optimal policy. To do this, let us examine three particular information states, each of which places all its probability weight on two states (s_2 and s_3). The starting information state for this POMDP places equal weight on states s_2 and s_3; this information state is $x_0 = \langle 0, 0.5, 0.5, 0, 0\rangle$. After taking the optimal action in this state (a_2), the agent is then informed as to which of the two possible initial states it is in, either $x_2 = \langle 0, 1, 0, 0, 0\rangle$ or $x_3 = \langle 0, 0, 1, 0, 0\rangle$.

Assume that the optimal policy can be expressed using a single vector for each action. Let q_1 be the vector associated with action a_1 and q_2 be the vector associated with a_2. In the optimal policy, $x_0 \cdot q_2 > x_0 \cdot q_1$ (a_2 is optimal from the initial state), $x_2 \cdot q_1 > x_2 \cdot q_2$ (a_1 is optimal from s_2), and $x_3 \cdot q_1 > x_3 \cdot q_2$ (a_1 is optimal from s_3). The first inequality can be rewritten

$$q_1[s_2] + q_1[s_3] < q_2[s_2] + q_2[s_3]. \tag{7.3}$$

The second two inequalities are equivalent to $q_1[s_2] > q_2[s_2]$ and $q_1[s_3] > q_2[s_3]$, which together imply $q_1[s_2] + q_1[s_3] > q_2[s_2] + q_2[s_3]$. But this directly contradicts Inequality 7.3; thus, the assumption that a single vector per action suffices is in error.

Representing the optimal policy using *two* vectors for action a_1 is trivial.

7.6.4 A Piecewise-linear-convex Q-learning Algorithm

A simple approach to learning a piecewise-linear convex Q function is to maintain a set of vectors for each action and to use a competitive updating rule: when a new training instance (i.e., information state/value pair) arrives, the vector with the largest dot product is selected for updating. The actual update follows the linear Q-learning rule. In some cases, different vectors will come to cover different parts of the space and thereby represent a more complex function than would be possible with a single vector [94].

Although this algorithm performs well on some problems, its performance on other problems has been disappointing. The primary difficulty is that noisy updates can cause a vector to "sink" below the other vectors. Since this approach only updates vectors when they are the largest for some information state, these sunken vectors can never be recovered. A related problem plagues almost all competitive learning methods [66].

A classic approach to the sunken-vector problem is to avoid hard "winner-take-all" updates. Parr and Russell [117] solved information-state MDPs using a differentiable approximation of the maximum function and found they could produce good policies for many simple POMDPs. The approach is promising enough to warrant further study.

7.7 Open Problems

Algorithms for solving information-state Markov decision processes are still being developed and many questions remain.

- Are there any provably efficient approximate solutions to the information-state MDP?

- There are value functions for POMDPs with n states and $2n$ actions where the number of vertices in a value-function region is 2^n. By analogy with existing work on counting the vertices of polyhedral regions [80], it ought to be possible to construct an example with a constant number of actions and a logarithmic number of observations for which the number of vertices in some region is exponential. Can such an example be identified?

- It is possible to find the best linear approximation (in a max norm sense) to a set of points using linear programming. Is there an extension of this result to more complex approximations? This would have implications for learning optimal value functions and putting bounds on the suboptimality of a learned value function.

- In this section, I suggested that the class of polynomially action-output bounded POMDPs was worthy of study. This comment was motivated by complexity-theoretic concerns. Are there naturally occurring POMDP subclasses that can be identified and explored?

- Are there POMDPs in which $|\Gamma_t|$ grows as a double-exponential function of t? Are there POMDPs in which $|\Gamma_t|$ grows a a single-exponential function of t? Is it ever the case that the PSPACE algorithm of Chapter 6 is superior to the algorithms described in this chapter?

- Although lexicographic ordering plays a crucial role in separating useful and non-useful policy trees, there is a sense in which it is an artifact of the proof of Lemma 7.1. Is there another simple way to quickly identify useful policy trees?

- Q-learning is known to converge to the optimal Q function in finite-state MDPs, under the right conditions. Reinforcement-learning methods do not appear to converge for general information-state MDPs. Is there some way of structuring the problem so it is solvable by reinforcement learning? Does linear Q-learning converge to the optimal Q functions if they are linear? Does linear Q-learning converge to an optimal policy if it is linearly representable? Schapire and Warmuth [136] showed that a minor variation of TD(λ) performs reasonably well provided that there is some linear predictor that performs well; could these results shed some light on learning good policies?

7.8 Related Work

Algorithmic approaches to solving POMDPs, especially in terms of the information-state MDP, have been surveyed extensively. The surveys by Monahan [109], Lovejoy [100], and White [176] are all clear, concise, and contain a great deal of useful information.

Although in this chapter I focus on approximating the infinite-horizon problem using exact methods for the finite-horizon, other methods have been explored. Lovejoy's survey cites several other methods for solving the information-state MDP in addition to those that use a piecewise-linear convex representation of the value function; for instance, in one class of methods, the infinite-horizon value function is approximated using a fixed grid of information states [99].

Sondik [150] presented a policy-iteration algorithm for finding approximate solutions to infinite-horizon POMDPs. Sawaki and Ichikawa [135] advocated the use of the value-iteration method, effectively reducing the problem of finding an approximate infinite-horizon value function to that of finding an exact finite-horizon value function. This is identical to the approach taken in this chapter.

The vector representation of finite-horizon value functions was first explored by Sondik [149] in his dissertation, which made it possible for a computational treatment of POMDPs to commence. The policy-tree representation is implicit in his work, and was made explicit by Cassandra, Kaelbling and Littman in the course of this research [32, 73].

Monahan [109] provided the first description of the enumeration method, also implicit in Sondik's work. Monahan attributes the enumeration algorithm to Sondik, although later authors [33, 100] give the credit to Monahan. Smallwood and Sondik's one-pass algorithm [148] avoids enumeration, at the expense of extensive record keeping. Cheng [33] developed a collection of POMDP algorithms and his Ph.D. thesis surveys almost all the algorithms existing at the time. White and Scherer [177] extended the reward-revision method, developed for MDPs, to POMDPs.

The development of the witness algorithm [32, 92, 95] was inspired most directly by Cheng's linear support algorithm [33], with the difference that standard linear programming was to be used in place of vertex enumeration to identify missing vectors. An early version was shown to be incorrect [92], and later versions introduced the idea of finding a representation for the Q functions. As presented here, the algorithm bears a close resemblance to Lark's filtering algorithm [176].

Chrisman [34] introduced the POMDP model to the reinforcement-learning community. His work, and that of McCallum [104], primarily addressed learning the POMDP model itself, and used the simplest possible representation for value functions.

Methods for solving the continuous state-space information-state MDPs that come from POMDPs must work with a parameterized representation of the value function. Except in some very special cases, these representations are approximate. Reinforcement learning and dynamic programming using approximate value functions is attracting increasing interest. Boyan and Moore [29] examined methods for solving a particular class of continuous state-space MDPs, Gordon [58] and Tsitsiklis and Van Roy [164] demonstrated closely related provably convergent dynamic-programming algorithms, and Baird [7] derived a gradient-descent rule for adjusting the parameters representing a value function in a reinforcement-learning setting; a survey of these techniques and others has recently been compiled [30].

The linear Q-learning and piecewise-linear convex Q-learning rules were developed in a parallel research effort by Littman, Cassandra, and Kaelbling [94]. Independently, Russell and Parr [117] attacked the same problem using a more complex value-function representation that can be adjusted by gradient descent.

7.9 Contributions

Information-state MDPs arise as a way of coping with the potentially unbounded histories that must be considered when solving partially observable Markov decision processes. Exact algorithms for solving information-state MDPs over finite horizon have been around for 25 years, though a careful complexity study of these algorithms had not been undertaken. I provide a new worst-case analysis of several algorithms for solving this problem, and explain that, even when the optimal value function is simple, these algorithms can take exponential time. I develop a new complexity result that shows that it is likely that this is an inherent difficulty with the problem, for it can be solved in polynomial time if and only if all problems in NP can be solved in randomized polynomial time. I describe a new algorithm, called the witness algorithm, which I developed in collaboration with Cassandra and Kaelbling, and prove that it has complexity-theoretic properties that make it extremely attractive.

In the process of developing this algorithm, I discovered the importance of breaking ties between policy trees using a lexicographic ordering, and developed efficient algorithms for doing so. The concepts I derived in this context are critical to bounding the run time of earlier algorithms as well, although, because this is the first in-depth analysis of these algorithms, this fact was not recognized.

In the area of reinforcement learning, I analyzed the replicated Q-learning rule and argued that the new linear Q-learning rule is more appropriate. I also provided a concrete example for which no linear representation will suffice to encode optimal behavior.

Although the witness algorithm goes a long way toward solving information-state MDPs efficiently, it is likely that no exact algorithm will be effective for solving large-scale POMDPs. Other methods that make use of value-function approximation or finite-memory policies appear more promising in the short run. It is my hope that the techniques presented in this chapter will inspire and inform the development of more practical approaches.

Chapter 8

Summary and Conclusions

The central thesis of this work is that designing algorithms with attention to complexity and convergence analysis can make it possible to solve larger and more difficult sequential decision-making problems. I illustrated this point by analyzing existing algorithms to indicate which hold the most promise for solving large problem instances, deriving complexity results to show which problems are unlikely to be solvable efficiently without additional restrictions, and inventing new algorithms with provably superior run times, wider coverage of problem instances, or guaranteed convergence.

This chapter provides several "big-picture" comparisons among the sequential decision-making models discussed throughout the thesis. It is intended to convey a feel for the state of the art in algorithms for sequential decision making, as well as to point the way to the next set of problems to be solved.

8.1 Comparison to Artificial Intelligence Planning

In this section, I provide an extremely brief summary of work in planning and relate it to the results I described.

8.1.1 Deterministic Environments

Planning in artificial intelligence is concerned with finding good behavior given a description of an environment. In traditional planning research, environments are deterministic and fully observable, and the objective is to find any sequence of actions that moves the agent from a prespecified start state to any of a prespecified set of goal states—to solve deterministic goal-oriented MDPs.

This model is the simplest type of sequential decision-making problem considered in this thesis. What makes it difficult and worthy of study is that the states of the environment are represented in a *propositional* form. Let us consider a simple example, adapted from a paper by Draper, Hanks, and Weld [50, 49].

The environment is a manufacturing plant and the agent's task is to process and ship a particular widget. At any moment in time, the widget is either painted (PA) or not, flawed (FL) or not, blemished (BL) or not, shipped (SH) or not, rejected (RE) or not, and the supervisor has either been notified (NO) or not. The actions available to the agent are to: REJECT the widget, which should happen if it is flawed; PAINT the widget, which it must do to process it; SHIP the widget, which it should do if the widget is processed; and NOTIFY the supervisor when processing is complete.

There are two possible initial states, FL BL $\overline{\mathrm{PA}}$ $\overline{\mathrm{SH}}$ $\overline{\mathrm{RE}}$ $\overline{\mathrm{NO}}$ and $\overline{\mathrm{FL}}$ $\overline{\mathrm{BL}}$ $\overline{\mathrm{PA}}$ $\overline{\mathrm{SH}}$ $\overline{\mathrm{RE}}$ $\overline{\mathrm{NO}}$. The objective is to end up in either of two goal states, FL BL $\overline{\mathrm{PA}}$ $\overline{\mathrm{SH}}$ RE NO or $\overline{\mathrm{FL}}$ $\overline{\mathrm{BL}}$ PA SH $\overline{\mathrm{RE}}$ NO. The shortest valid plan is REJECT NOTIFY if the part is flawed or PAINT SHIP NOTIFY if the part is not flawed. Traditional planners have very little trouble finding these plans.

As there are six propositions and each can take on two values, the effective state space consists of $2^6 = 64$ states. In general, the size of the state space is exponential in the number of propositions. In addition, the transitions and rewards can often be specified compactly in terms of the propositions themselves, so a complete description of the domain can be made significantly smaller than an exhaustive listing of the components of the T and R functions. This means that the fastest possible algorithm for solving deterministic MDPs will always be exponential, simply because it must consider each of the states independently. The challenge of traditional planning research is to find algorithms that can run efficiently with respect to the size of the compact representation of the environment.

8.1.2 Stochastic Environments

In spite of the difficulty of compact planning problems, researchers have begun to reach the limits of problems that can be solved usefully as deterministic goal-oriented problems. Work in decision-theoretic planning has broadened its scope to stochastic environments with more elaborate reward structures. The fundamental difference between work in operations research on MDPs and the work in artificial intelligence on decision-theoretic planning is the representation of states and actions; planning researchers work with compact propositional representations of the state space, while MDP researchers assume a collection of unanalyzed, independent states.

Of course, this difference in assumptions translates to substantial differences in the types of algorithms that can be used to find optimal behavior; one fundamental difference is in the form of the output of the algorithms.

Let us restrict our attention for the moment to stochastic, goal-oriented environments. We would like to know how to behave over the finite or infinite horizon so that the probability of reaching the goal from some start state is at least $1-\delta$. The output of an MDP algorithm for this problem would be an optimal policy that lists, for each state, the best choice of action. This is impractical for planning problems, however, because the number of states is too large. In deterministic environments, the solution to this is to return a *plan*, or sequence of actions, instead of a policy. The size of an optimal plan is a function of the number of steps needed to reach the goal, not the size of state space, so it can be a very efficient representation.

In a stochastic environment, a linear plan is not sufficient to describe optimal behavior[1]; it is necessary to conditionalize the actions on the results of uncertain transitions. The simplest extension over a linear plan is a plan tree, which is an agent's course of action that includes "forks" that occur when an action can have multiple outcomes. Environments that are generally deterministic can have good plan trees that are small, and planning algorithms have been developed that can find these plan trees.

As the number of possible action outcomes grows, the branching factor for the associated plan tree grows as well. It rapidly becomes inefficient for a plan to associate each possible sequence of outcomes with an action. It makes more sense to map states to actions than to construct a full plan tree. Such plans have been called "universal plans" [138] and are equivalent to what I have been calling "policies."

I am not aware of any detailed comparisons of MDP and planning algorithms in uncertain domains. I understand that MDPs with hundreds of thousands of states can be solved using current approaches, which translates to problems with perhaps 17 propositions. It is my opinion that a combination of techniques from operations research and artificial intelligence will be needed to solve large planning problems efficiently; this

[1]Kushmerick et al. [87] attempt to find good linear plans for stochastic environments.

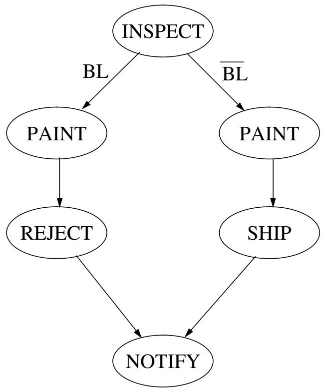

Figure 8.1: Plan for a partially observable environment.

issue is explored by Boutilier, Dean, and Hanks [23].

8.1.3 Partially Observable Environments

Several projects have concerned themselves with planning in partially observable environments. Plans for partially observable environments cannot be conditioned on the underlying state, i.e., the value of the propositions. Instead, actions are endowed by the environment with "observational effects" and these effects (noisily) reveal the values of particular propositions.

We can extend the widget-processing example above by making the two possible starting states equally likely and by introducing an additional action, INSPECT, which returns either "blemished" or "not blemished" depending on the state of the BL proposition. Figure 8.1 gives a simple DAG-structured plan that reaches a goal with high probability.

Very few algorithms have been proposed for this type of problem. Of course, any of the POMDP algorithms described in Chapter 6 can be used, once the complete state space is constructed. The C-Buridan [50, 49] and structured policy iteration [25] algorithms solve partially observable problems using compact representations directly. The C-Buridan algorithm discovers the DAG-structured plan in Figure 8.1. Draper et al. note that "significant search control knowledge is necessary to enable solution of even simple examples" like the widget-processing example. This means that C-Buridan is unlikely to have found the plan in Figure 8.1 in any reasonable amount of time without additional information about the environment.

In contrast, the witness algorithm performed quite well on this problem. Of all possible three-step plans, it found one with the maximum success probability in a few seconds using no additional information about the domain. This result illustrates two points. First, the witness algorithm can be used to solve compactly specified planning problems competitively with state-of-the-art planning algorithms. Second, the state of the art in planning in POMDPs is still quite primitive. I personally believe that the witness algorithm is not the best way to solve compact POMDPs in general. However, I believe that insights from the witness algorithm could make important contributions to algorithms for planning in partially observable stochastic domains.

In conclusion, I believe that the algorithms described in this thesis are quite relevant to planning researchers because (a) they can be used as alternatives to existing planning algorithms, but more importantly (b) there are important insights here that can probably be used to complement existing planning algorithms to solve bigger and more complex problems.

8.2 Comparison of Game Models

Table 8.1 compares and contrasts MDPs, alternating Markov games, and Markov games using the results from Chapters 2, 4, and 5. The first row summarizes the long-term behavior of value iteration for each model. For both MDPs and alternating Markov games, value iteration identifies an optimal policy in a pseudopolynomial number of iterations (Sections 2.4.2 and 4.4.1). For Markov games, value iteration generates policies that become closer and closer to optimal but never necessarily get there (Section 5.4.2). The second row summarizes the result of a linear-programming approach to solving these models. Whereas linear programs can be used to solve MDPs (Section 2.3.3), it is not known whether alternating Markov games can be solved this way (Section 4.4.3). However, for general Markov games, it is possible to argue that no linear program can be used to identify the optimal policy (Section 5.4.3).

The third row describes the form of the optimal policy: for MDPs and alternating Markov games, a stationary deterministic policy suffices, while, for general Markov games, it is necessary to consider probabilistic policies as well. The fourth row gives the state of our understanding of the computational complexity of solving the three models: MDPs can be solved in polynomial time (Section 2.5); alternating Markov games are not known to have a polynomial-time solution, but belong to the class NP$\cap$co-NP, providing some evidence that a polynomial-time algorithm exists (Section 4.5); and

	MDPs	alternating Markov games	Markov games
value iteration	optimal policy	optimal policy	approaches optimal
naive linear program	solves	fails	impossible
optimal policy	deterministic	deterministic	stochastic
computational complexity	polynomial	NP$\cap$co-NP	irrational
reinforcement learning	approaches optimal	approaches optimal	approaches optimal

Table 8.1: Comparison of properties of Markov decision processes, alternating Markov games, and Markov games.

optimal solutions to Markov games may involve irrational numbers even when the rewards and transitions are rational, making it unlikely that any algorithm can solve them exactly (Section 5.5). The fifth and final row states that reinforcement-learning algorithms exist for all three models that result in arbitrarily good approximations to the optimal policy under the proper conditions.

It is worth noting that every MDP is an alternating Markov game and every alternating Markov game is a Markov game, and thus any algorithm for solving Markov games can be used to solve MDPs as well.

8.3 Complexity Summary

There are many aspects by which the value of an algorithm can be judged, but without either theoretical analysis or empirical evaluation, the development of algorithms is an exercise in aesthetics: Can you find an algorithm that is sufficiently "elegant"?

Of course, sound theoretical and empirical study of an algorithm is difficult. Which example problems are the most revealing empirically? Do the assumptions necessary to make for the analysis to work hold in practice? How can we be sure our empirical results generalize? Are the constants in our asymptotic analysis small enough to affect the run time of a small to medium-size problem instance?

One major challenge in any theoretical analysis of an algorithm is to determine whether the upper and lower bounds on the run time are "tight", that is, whether they accurately characterize the range of possible run times of the algorithm. If the analysis *is* tight, one needs then determine whether some other algorithm might produce faster run times. These issues can be extremely difficult to address.

	polynomial horizon	infinite horizon
deterministic MDP	NC [116]	NC [116]
MDP	P-complete [116]	P-complete [116]
deterministic alternating Markov game	P-complete*	NP∩co-NP [183], P-hard*
alternating Markov game	P-complete*	NP∩co-NP [36], P-hard*
deterministic Markov game	P-complete*	irrational*, P-hard*
Markov game	P-complete	irrational [169], P-hard*
deterministic unobservable MDP	NP-complete*	PSPACE-hard*/ EXPTIME*
unobservable MDP	NP-complete [116]	PSPACE-hard*
deterministic POMDP	NP-complete*	PSPACE-hard*/ EXPTIME*
POMDP	PSPACE-complete [116]	EXPTIME-hard*

Table 8.2: Summary of complexity results for finding optimal policies.

Complexity theory can be invaluable in determining when algorithmic development and analysis need to change direction. Complexity analysis addresses the inherent difficulty of the problem that is being solved. Finding that infinite-horizon POMDPs are EXPTIME-hard immediately implies that your algorithm for solving them will *not* run in polynomial time—no further analysis is needed. In addition, if you do find an algorithm that runs in exponential time, you need not bother searching for an algorithm with better worst-case run time; at this point, it is better to focus on algorithms for special cases, algorithms that run well on "average" problems, or some other less traditional avenue of algorithm development.

Table 8.2 summarizes the complexity results for the various models. Results marked with asterisks were first proven in this thesis.

A definite trend is visible in this summary: increasing the amount of uncertainty in a problem increases the complexity of the problem. An agent in a infinite-horizon POMDP environment faces an uncertain future because of stochastic transitions, an uncertain state because of stochastic observations, and an indefinite horizon. The corresponding computational complexity is extremely high. An agent in a deterministic Markov game environment faces a different type of uncertainty: uncertainty about the agent's behavior and its current choice of action. Again, the complexity is high, but in a different way.

These results indicate that, wherever possible, it is important to eliminate uncertainty from applications. Nevertheless, when significant uncertainty is present, algorithms are available that find near-optimal behavior.

8.4 Contributions

Although much of the thesis was devoted to summarizing and unifying results from a number of different disciplines, I also presented the novel results listed below.

- Markov decision processes
  - proof that policy iteration runs in pseudopolynomial time, and in polynomial time for any fixed discount factor
  - generalized convergence proof for Q-learning
  - simplified proof of the convergence of value iteration for all-policies-proper MDPs
  - demonstration that the deterministic case can be viewed in the closed semiring framework
- Generalized Markov decision processes
  - introduction of a new model
  - proof of convergence of value iteration
  - proof of convergence of policy iteration
  - proof of convergence of model-free reinforcement learning
  - proof of convergence of model-based reinforcement learning
- Markov games
  - reinforcement-learning method (minimax-Q)
  - proof of convergence of reinforcement-learning methods
  - proof of convergence of self-play approaches
  - demonstration that the deterministic case can lead to irrational values

  - polynomial-time algorithm for a special case: constant reward-cycle alternating Markov games
  - proof of P-hardness for the deterministic finite-horizon case

- Partially observable Markov decision processes

  - model-based algorithm (witness)
  - method for learning immediate rewards in an unknown model
  - proof that the deterministic polynomial-horizon case is NP-complete
  - proof that the deterministic infinite-horizon case is PSPACE-hard, in EXPTIME, and PSPACE-complete if rewards are boolean
  - proof that the stochastic infinite-horizon case is EXPTIME-hard, and EXPTIME-complete if rewards are boolean
  - proof that a single step of value iteration using the policy-tree representation is NP-hard under randomized reductions
  - analyses of several existing algorithms for solving POMDPs.

8.5 Concluding Remarks

Sequential decision making is one of the most important problems in artificial intelligence and perhaps all of computer science, and the search for efficient algorithms is still relatively young. The most important areas of future research are (1) finding restrictions to some of the harder problems that capture the structure of problems in the real world while admitting efficiently computable approximate solutions, and (2) finding efficient algorithms for models specified in structured form.

Both of these areas will require significant synergy between researchers exploring applications and researchers inventing and analyzing new algorithms. As long as the lines of communication remain open, there is every reason to be optimistic that the coming years will bring more efficient algorithms for more important problems.

Appendix A

Supplementary Introductory Information

In this appendix, I provide background material on computational complexity and linear programming, and summarize some of the conventions I use to illustrate the algorithms in this thesis.

A.1 Computational Complexity

The goal of research into *computational complexity* is to classify and categorize computational problems. The most fundamental quantity in this area is run time: How much time does it take to solve a particular problem? At the same time, it is often useful to constrain other quantities such as the space used or the number of processors required in a parallel implementation.

Time is measured in terms of basic computer operations like simple logical operations or branches. In the *arithmetic model*, arithmetic operations are assumed to take unit time. In the *bit-operation model*, the time taken for arithmetic operations is a function of the magnitude of the numbers involved. Since we expect the numbers used in the programs to be represented using conventional computers, we most often restrict our attention to integers or rational numbers (the ratio of two integers). The arithmetic model is often more intuitive to work with and when we can guarantee that all operations will be on rational numbers of a given precision, it is easy to convert bounds under an arithmetic model to those under a bit-operation model.

Most complexity results are framed in terms of *decision problems*—problems that require a yes/no response for each well formed input. The time and space complexity of decision problems are given as a function of the amount of space needed to write down an input. The goal of complexity analysis is to find, for a given class of decision problems, how the worst-case run time of an algorithm scales as a function of the input length. Note that many fundamental questions in this area are open. This means that many important results are given in the slightly cumbersome form "such and such is true only if some complexity theoretic property which is generally believed to be true is true." I examine examples of this in the next section.

A.1.1 Complexity classes

Here are brief descriptions of some of the complexity classes I use in this thesis.

- P: The set of decision problems that can be answered with certainty in a polynomial number of operations (polynomial time). Traditionally, this class has been equated with the set of problems that have efficient algorithms.

- NP: The set of decision problems that can be answered non-deterministically in polynomial time. Most NP problems have the property that a "yes" answer can be supported by a short example.

- co-NP: This is the complement of NP. Most co-NP problems have the property that a "no" answer can be supported by a short example.

- NP∩co-NP: This is the set of problems that are in both NP and co-NP. Membership in this class can be taken as evidence that a problem is in P [55], although there are some significant examples in this class whose exact complexity remains unknown.

- NC: The set of problems that can be decided on a parallel computer using a polynomial number of processors and polylogarithmic ($O(\log^k n)$, for some k) time. All problems in NC are in P.

- P-hard: The set of problems such that *if* they are in NC, all problems in P are in NC.

- RP: The set of problems that can be decided in randomized polynomial time, that is, if the answer is truly "yes," an algorithm will give a "yes" answer with at most a bounded probability of error; if the answer is "no," the algorithm will say "no." All decision problems in P are in RP and all decision problems in RP are in NP.

- PSPACE: The set of problems that can be decided using polynomial space. All problems in P and NP and co-NP are in PSPACE.
- EXPTIME: The set of problems decidable in exponential time. All problems in PSPACE are in EXPTIME.
- NP-hard: The set of problems such that *if* they are in P, all problems in NP are in P. Similarly for PSPACE-hard.
- NP-complete: Problems in NP that are NP-hard. They are the hardest problems in NP and many natural decision problems are in this class. No efficient algorithms are known for exactly solving problems in this class. The important and open question "does P=NP?" basically boils down to whether any NP-complete problem can be solved in polynomial time. For our purposes, we assume that P mostly likely does not equal NP and therefore that any NP-complete or NP-hard problem is intractable in general.
- PSPACE-complete: Analogous to NP-complete. Even if P=NP, it is very likely that PSPACE does not equal P. Showing that a problem is PSPACE-complete is very nearly a proof of its intractability in general.
- EXPTIME-complete: The hardest problems in EXPTIME. Includes problems that are provably intractable, therefore the class of EXPTIME-complete problems is the easiest set of problems known to be intractable.

Saying a problem is in PSPACE (or NP or EXPTIME or NC or RP) gives an upper bound on its difficulty, since knowing that a problem is in PSPACE, for example, means that we never need more than polynomial space to solve it. Saying a problem is NP-hard (or PSPACE-hard or EXPSPACE-hard or P-hard) gives a lower bound on its difficulty since an NP-hard problem is no easier than any problem in NP. "Completeness" results are particularly tidy since they give matching upper and lower bounds on the difficulty and as such (with regard to current theory) exactly determine how hard a problem is. For more background information on decision problems and NP-completeness, see Garey and Johnson's book [55].

A.1.2 Reductions

A *reduction* is simply a mapping of an instance of one problem A to an instance of another B, so that the solution to B can be used to solve A. Reductions are extremely important for relating two problems to show that one is no easier than the other. For example, by reducing A to B, we show that solving A is no more difficult than solving B. And if the transformation from A to B and from the answer for B back to the answer for A takes, for instance, polynomial time, then A takes no longer to solve than B to within a polynomial factor.

One use of reductions is to show that a given problem is hard. An example, discussed in more detail in Chapter 6, is that solving finite-horizon POMDPs is PSPACE-hard. The proof goes like this. Take an instance of the *quantified-boolean-formula* problem. It is known that the quantified-boolean-formula problem is PSPACE-hard (PSPACE-complete, in fact), meaning that a polynomial-time algorithm for solving the problem in general would prove that all problems in PSPACE are solvable in polynomial time. It is possible to show that a finite-horizon POMDP can be constructed from an instance of the quantified-boolean-formula problem in polynomial time, and that the solution to the POMDP could be used to provide a solution to the quantified-boolean-formula problem in polynomial time. Since this is true, a polynomial-time algorithm for solving general finite-horizon POMDPs would provide a polynomial-time algorithm for the quantified-boolean-formula problem, which would, in turn, imply that all problems in PSPACE are solvable in polynomial time. Thus the finite-horizon POMDP problem is PSPACE-hard.

It is also possible to use the notion of reductions to show how easy a given problem is. An example of this is in Chapter 2, where the problem of solving MDPs is reduced to the problem of solving linear programs. Since the reduction can be carried out in polynomial time, and since linear programs can be solved in polynomial time, this proves that MDPs can be solved in polynomial time. Interestingly, this is the only existing proof of this fact.

In the next section, I use reductions to show that many decision problems are equal in difficulty to their corresponding optimization problems.

A.1.3 Optimization Problems

Despite the mathematical richness of decision problems, most naturally occurring problems do not appear in a "yes/no" form, and hence the complexity classes described above cannot really be applied to them. Nevertheless, many optimization problems, such as the sequential decision-making problems that are the topic of this thesis, are *polynomially reducible* to decision problems.

For example, consider the following problem: given a description of an MDP, find the optimal policy. A related decision problem might be: given a description of an MDP, and a rational number w, and a starting state s_0, is there a policy that achieves expected reward greater than or equal to w starting from s_0?

Clearly if we could solve the optimization problem, the decision problem would be trivial—simply take the MDP, find its optimal policy, and evaluate it to see if $V^*(s_0) \geq w$.

We can also use an efficient solution to the decision problem to solve the optimization problem. The basic idea is to use binary search to find the largest possible value for w for each s_0. As long as we have a guarantee that the optimal value function consists of polynomial-precision rational numbers (and we do for MDPs, see Theorem 2.1), a polynomial-time answer to the decision problem gives a polynomial-time answer to the optimization problem.

Thus, for this and many other problems, the decision-problem formulation and the optimization formulation are "equivalent" in that they have the same worst-case run time to within a polynomial factor. For many problems, it is therefore reasonable to use the decision-problem terminology when taking about optimization problems.

These reductions almost always depend on the solution to the optimization problem being a rational number expressible with at most a polynomial number of bits. Thus, showing that an optimization problem can have an irrational solution, even for a problem specified with only rational values, is taken as evidence that the optimization problem is strictly harder than its corresponding decision problem.

A stronger result might be to show that solving the optimization problem is equivalent to finding exact solutions to arbitrary polynomial equations. Galois theory tells us that there is no finite-time algorithm (restricted to simple arithmetic operations and roots) for solving arbitrary polynomial equations [6]. Thus, such problems are, in a sense, *uncomputable*. In spite of the difficulty of this class of optimization problems, several of them admit ϵ-optimal approximations that can be computed in time that is a function of $1/\epsilon$.

A.1.4 Other Complexity Concepts

There are several other complexity-theoretic concepts that are used in this thesis. They involve the specification of a problem's complexity when the input involves representations of numbers.

When a problem is in P, it has an algorithm whose run time scales as a polynomial in the size of the input. When the input involves numbers, we assume that these numbers are given in the most compact possible form, for example, as integers specified in binary notation. Similarly, when we say a problem is NP-complete, we are stating that it is hard with respect to a compact representation of the input numbers.

Sometimes it is interesting to consider the complexity of the problem when its input is given in unary. This means we are considering the run time with respect to the *magnitude* of the numbers involved and not the size of their representations in binary. Presenting the input numbers in unary makes a problem easier in the following sense. Consider the problem of determining if a number x is prime. One simple algorithm checks every number from 2 to $\lfloor\sqrt{x}\rfloor$ to see if any divide x evenly. The run time of this algorithm is proportional to $\lfloor\sqrt{x}\rfloor$ and is thus polynomial in the input size if x is expressed in unary, but exponential in the input size if x is expressed in binary.

With this in mind, here are some major categories of time complexity:

- strongly polynomial: The run time of the algorithm, measured in arithmetic operations, can be bounded as a polynomial function of the size of the input, measured as the number of numbers. An example is matrix multiplication which needs no more than n^3 arithmetic operations to multiply two n by n matrices.

- polynomial: The run time of the algorithm, measured in arithmetic or bit operations, can be bounded as a polynomial function of the size of the input, measured in bits. Although all strongly-polynomial algorithms are polynomial, some polynomial-time algorithms (such as the ellipsoid method for solving linear programs and MDPs) are not strongly polynomial because the number of arithmetic operations needed depends on the size of the numbers involved.

- pseudo-polynomial: The run time of the algorithm, measured in arithmetic or bit operations, can be bounded as a polynomial function of the magnitude of the numbers in the input. An example is the value-iteration algorithm for solving MDPs, for which the number of iterations can grow as a polynomial function of the magnitude of the discount factor, $1/(1-\beta)$. Any polynomial-time algorithm is also pseudo-polynomial.

- NP-complete: A decision problem is NP-complete if it is in NP and the existence of a polynomial-time algorithm to solve it would imply that P=NP.
- strongly NP-complete: A decision problem is strongly NP-complete if it is in NP and the existence of a pseudo-polynomial-time algorithm to solve it would imply that P=NP.

A.2 Algorithmic Examples

Throughout this thesis, I use short code fragments to make concrete the algorithms being discussed. I use a number of conventions to help make the code fragments as clear and simple as possible.

First of all, the notation I use does not correspond to any existing computer language. It borrows a great deal of structure from "C," but with a number of extensions for more complex data types.

Table A.1 contains a meaningless subroutine that illustrates some of the conventions used to define program fragments. In this example, a subroutine `greeting` is defined to take 3 arguments, a, b, and c. The local variable t is initialized to zero and then doubled and incremented by k, for each k from 1 to a. Next, a is increased by 3 if t does not equal b and c is greater than or equal to 10. Otherwise, b is raised to the second power and c is assigned the value of $b-3$. Finally, the subroutine returns the largest value of a, b, and c.

The distinction between global and local variables should be clear depending on the context. The data type of a given variable is not given explicitly, but again, it should always be clear from the context or the associated text.

Typographically, names of user-defined subroutines are in typewriter font, reserved words are bold, variables are in italics, and mathematical functions are in roman font.

For the most part, any subroutine called within the definition of another subroutine will be defined elsewhere in the thesis. The only difficult functions that are left unspecified are routines for solving linear programs and systems of linear equations. The first is intended to take a set of variables, linear constraints on those variables, and an objective function, and return bindings for the variables that satisfy the constraints and

$\texttt{greeting}(a, b, c) := \{$
    $t := 0$
    foreach $k \in [1 \ldots a]$ $t := 2t + k$
    if $((t \neq b)\text{and}(c \geq 10))$ **then** $a := a + 3$
    else $\{$
        $b := b^2$
        $c := b - 3$
    $\}$
    return $\max\{a, b, c\}$
$\}$

Table A.1: Example subroutine illustrating the sample-code conventions used throughout this thesis.

maximize the objective function. The second finds a binding for a set of variables that satisfies a set of equality constraints on linear functions of those variables. Although neither of these procedures are considered standard in most programming languages, there are many excellent commercial (and free) software packages available.

My intention is that an experienced programmer should have little trouble implementing the algorithms described in this thesis.

A.3 Linear Programming

Nearly every chapter refers to, or makes use of, linear programming. Very briefly, a linear program consists of a set of variables, a set of linear inequality constraints on those variables, and a linear objective function to be either maximized or minimized.

Linear programming is interesting because it is one of the most difficult and general problems that can be solved in polynomial time. The first theoretically efficient algorithm for solving linear programs, the ellipsoid algorithm [79], does not appear to be of practical use; however, refinements of Karmarkar's [78] polynomial-time algorithm are competitive with the fastest practical algorithms. Another algorithm for solving linear programs, the *simplex method* [41], is theoretically inefficient but runs extremely quickly in practice.

An excellent book by Schrijver [140] describes the theory of linear programs and the algorithms used to solve them.

Appendix B

Supplementary Information on Markov Decision Processes

In this appendix, I present an analysis of two results from Puterman's MDP textbook [126] and discuss their implications for the complexity of MDPs. I also prove that deterministic MDPs are closed semirings.

B.1 Comparing Policy Iteration and Value Iteration

This result is given much more precisely in Puterman's MDP book [126] as Theorem 6.4.6. The goal of this section is to provide intuitive verbal arguments so that the critical points can be examined more closely. We start with a host of definitions.

Let π_0 be an arbitrary policy. For $t \geq 0$, let V_t be the value function for π_t and π_{t+1} be a greedy policy with respect to V_t. Thus, the V_t functions form a sequence of value functions obtained by executing policy iteration starting from π_0. Let U_0 be the value function for π_0 (i.e., V_0) and for $t \geq 0$, let U_{t+1} be the result of applying value iteration to U_t. Thus the U_t functions form a sequence of value functions obtained by executing value iteration starting from π_0's value function. Let V^* be the optimal value function.

Theorem B.1 *For all $s \in S$ and $t \geq 0$, $U_t(s) \leq V_t(s) \leq V^*(s)$, and therefore policy iteration converges no more slowly than value iteration (i.e., at least linearly).*

Proof: The easy part is showing that $V_t(s) \leq V^*(s)$ for all $s \in S$ and all $t \geq 0$. This follows from the fact that V_t is the value function for a particular policy and the optimal value function is larger than all such value functions at all states.

To show that $U_t(s) \leq V_t(s)$ for all $s \in S$ and $t \geq 0$, we proceed by induction on t. For $t = 0$, clearly $U_0(s) \leq V_0(s)$ for all $s \in S$ since U_0 is defined to be equal to V_0. This proves the base case.

Let us assume that $U_t(s) \leq V_t(s)$ for all $s \in S$. We will use this to show that $U_{t+1}(s) \leq V_{t+1}(s)$ for all $s \in S$.

First, let U'_{t+1} be the value function that results from taking one step of value iteration on V_t. I will argue that $U'_{t+1}(s) \leq V_{t+1}(s)$ and that $U_{t+1}(s) \leq U'_{t+1}(s)$ for all $s \in S$ and the inductive proof will follow from chaining these inequalities.

Note that U'_{t+1} is the value function for a non-stationary policy that follows π_{t+1} for one step and then π_t thereafter. This non-stationary policy is no worse than one that follows π_t forever since the non-stationary policy chooses its first action to maximize the long term reward *given* that π_t will be followed thereafter. Certainly this is no worse than following π_t all along. Thus $U'_{t+1}(s) \geq V_t(s)$ for all $s \in S$.

The quantity $U'_{t+1}(s) - V_t(s) \geq 0$ is the amount of improvement the non-stationary policy achieves over policy π_t starting from state s. Following policy π_{t+1} is like using the non-stationary policy but restarting it after each transition. This can be no worse than the non-stationary policy since it is as if the $U'_{t+1}(s) - V_t(s)$ gain is reaped on every step instead of just once. (This is true, although the actual argument is a bit more subtle. It can be proven as a consequence of the results in Section 3.3.3.) Since V_{t+1} is the value function for this policy, we have $V_{t+1}(s) \geq U'_{t+1}(s)$ for all $s \in S$.

How does U'_{t+1} compare to U_{t+1}? Well U'_{t+1} is the result of taking one step of value iteration on V_t and U_{t+1} is the result of taking one step of value iteration on U_t. Using the inductive hypothesis that $U_t(s) \leq V_t(s)$ for all $s \in S$, it is easy enough to show that $U_{t+1}(s) \leq U'_{t+1}(s)$ for all $s \in S$.

Chaining these results gives us the desired answer that $U_{t+1}(s) \leq V_{t+1}(s)$ for all $s \in S$ and therefore that policy iteration converges no more slowly than value iteration when started from the same point. $\square$

One extremely important observation is that this argument depends on the fact that policy updates are done in parallel. The proof does not hold for sequential improvement variations such as simple policy iteration. The reason is that, in comparing value iteration and policy iteration, the proof uses the fact that applying a step of either algorithm involves finding the same greedy policy with respect to the value function from the previous iteration. In value iteration, this greedy policy is adopted for a single step. In policy iteration, it is adopted as a stationary infinite-horizon policy. But deep down, it is the same policy and therefore the two algorithms have some common ground on which to be compared.

It is possible to imagine using Puterman's approach to compare other pairs of algorithms. For instance, consider a version of policy iteration in which on the tth iteration, the action choice for state $t \bmod |S|$ is flipped. It might be possible to compare this type of "single flip" policy-iteration algorithm to a version of value iteration where the value for state $t \bmod |S|$ is modified on step t. Unlike simple policy iteration, it is impossible for these algorithms to spend exponential time between considering changes to the action choice for any given state; it is likely that these algorithms will have similar convergence properties to true value and policy iteration.

On the other hand, it is difficult to imagine how this type of proof could be applied to simple policy iteration. In simple policy iteration, the state updated depends on the current value function. A related version of value iteration might be one in which the state with the lowest index number whose Bellman residual is nonzero is updated. I could easily believe that such a value-iteration algorithm would have terrible convergence properties and would therefore serve as a useless bound on simple policy iteration.

In summary, Puterman's proof relating value iteration to policy iteration makes use of the fact that both perform updates with respect to greedy policies. As a result, only parallel improvement policy iteration is covered by this theorem. Variations of policy iteration can be discussed, but only as they relate to analogous variations of value iteration.

B.2 On the Quadratic Convergence of Policy Iteration

Puterman [126] proves a theorem concerning the rate of convergence of policy iteration. It shows that, under the appropriate conditions, policy iteration converges at a rate that is quadratic (i.e., the error is squared on each iteration). The linear convergence of policy iteration [126] can be used to show that policy iteration runs in pseudopolynomial time (see Section 2.4.3). A proof of quadratic convergence would have even more important implications to the complexity analysis of policy iteration, so it is worth understanding what the theorem implies in this case. I will argue that Puterman's theorem applied to general finite state/action MDPs leads to a vacuous conclusion.

Here, P_π is the transition probability matrix associated with policy π.

Theorem B.2 *Suppose V_t is the tth value function generated by policy iteration, and that π_t is a greedy policy with respect to V_t, and there exists a $K, 0 < K < \infty$ for which*

$$||P_{\pi_t} - P_{\pi^*}|| \leq K||V_t - V^*|| \tag{B.1}$$

for $t = 1, 2, \ldots$. Then

$$||V_{t+1} - V^*|| \leq K\frac{\beta}{1-\beta}||V_t - V^*||^2. \tag{B.2}$$

Proof: See Puterman [126], Theorem 6.4.8. □

I begin by considering the conditions of the theorem, particularly Inequality B.1. First of all, as there may be more than one optimal policy, it is not necessarily the case that $||P_{\pi_t} - P_{\pi^*}||$ goes to zero as n increases. However, the theorem gives us the flexibility to choose π_t as any policy that is greedy with respect to V_t, so we choose π_t to be as similar as possible to π^*.

What more can we say about the convergence of the transition matrices in Inequality B.1? Convergence of V_t to V^* takes place in a finite number of iterations, which we call t^*. For $t \geq t^*$, we have equality in Inequality B.1 for all K, since both sides are equal to zero. For $t < t^*$ in deterministic MDPs, all the entries in the transition matrices for policies π_t and π^* are zeros and ones and the two matrices differ in at least one component. Therefore, for deterministic MDPs, $||P_{\pi_t} - P_{\pi^*}|| = 1$ for $t < t^*$ and 0 afterwards. Because deterministic MDPs are a subset of the MDPs to which this theorem should apply, it is important to see how the theorem applies to this case.

How do we choose K so that Inequality B.1 holds for all $t \geq 1$? I argued that the left-hand side of Inequality B.1 goes to zero in one discrete jump at t^*. In contrast, we know that $||V_t - V^*||$ goes to zero in a series of steps. Tseng [162] argues that when the rewards and transition probabilities of an MDP are all rational numbers, and V_t is the value function for some policy, then there is a value $\epsilon > 0$ such that $||V_t - V^*||$ is either zero or greater than or equal to ϵ. That is, any pair of value functions derived from policies that are closer than ϵ to one another are, in fact, exactly equal. This result provides a range of values for K that makes Inequality B.1 hold: $K \geq 1/\epsilon$. This works because the smallest possible value of $||V_t - V^*||$ for $t < t^*$ is ϵ, so $K||V_t - V^*|| \geq 1$, as required. In addition, any smaller value for K would not work, because ϵ represents the closest that the value functions for two policies can be without being equal.

I showed, at least at an abstract level, a way of satisfying Inequality B.1 by setting K appropriately, and gave an argument that smaller values of K will not suffice. Theorem B.2 relates the distance between the $(t+1)$-step value function and the optimal value function to the distance between the t-step value function and the optimal value function. For Inequality B.2 to be useful in proving the convergence rate of policy iteration, it must be that successive value functions are getting closer to optimal. Therefore, we need $||V_{t+1} - V^*|| < ||V_t - V^*||$, or

$$||V_t - V^*|| > K\frac{\beta}{1-\beta}||V_t - V^*||^2 \tag{B.3}$$

for $t < t^*$. Using the facts that $||V_t - V^*|| \geq \epsilon$, and $K = 1/\epsilon$, Inequality B.3 implies that $\beta < 1/2$. Or, to put it another way, when the discount factor is one half or more, the convergence bound given by Theorem B.2 allows the distance between value functions and the optimal value function to *grow* over successive iterations. This implies that Theorem B.2 is mute on the convergence rate of policy iteration unless $\beta < 1/2$. Section 2.4.3 shows that policy iteration, when the discount factor is bounded away from one, runs in polynomial time, so it appears that Theorem B.2 contributes little to the analysis.

Theorems related to Theorem B.2 are presented by Puterman and Brumelle [124, 125] in a more abstract setting that might make it possible to prove superlinear convergence of policy iteration, given some additional analysis.

B.3 Deterministic MDPs as Closed Semirings

In this section, I show how to define deterministic MDPs as closed semirings. As a consequence, deterministic MDPs can be solved in $O(|\mathcal{S}||\mathcal{A}| + |\mathcal{S}|^3)$ time.

Let $\mathbb{V} = \mathbb{R} \cup \{-\infty, +\infty\}$ and $\mathbb{L} = \mathbb{N} \cup \{-\infty, 0, +\infty\}$. Define $\mathbb{S} = \mathbb{V} \times \mathbb{L}$. An element of $\mathbb{S}$ is a summary of the discounted finite-horizon value of a path, where the first component gives the value and the second component gives the length. Note that $\mathbb{L} \subset \mathbb{V}$. Define $-\infty + v = -\infty$ for all $v \in \mathbb{V}$ and $+\infty + v = +\infty$ for all $v \in \mathbb{V} - \{-\infty\}$. Otherwise, $v_1 + v_2$ is defined as normal addition. For discount factor $0 < \beta < 1$, define $\beta^{-\infty} = +\infty$, $\beta^0 = 1$, $\beta^{+\infty} = 0$, and otherwise β^ℓ is defined as normal exponentiation for $\ell \in \mathbb{L}$. Finally, define $(+\infty)(-\infty) = -\infty$, $(+\infty)0 = 0$, $(+\infty)v = +\infty$ (for $v \in \mathbb{V} - \{-\infty, 0\}$), and otherwise $v_1 v_2$ is defined as normal multiplication.

Define operator $\oplus$ as follows. Let $(v_1, \ell_1) \in \mathbb{S}$ and $(v_2, \ell_2) \in \mathbb{S}$. Then

$$(v_1, \ell_1) \oplus (v_2, \ell_2) \equiv \begin{cases} (v_1, \ell_1), & \text{if } v_1 > v_2, \\ (v_2, \ell_2), & \text{if } v_1 < v_2, \\ (v_1, \ell_1), & \text{if } v_1 = v_2 \text{ and } \ell_1 > \ell_2, \\ (v_2, \ell_2), & \text{otherwise.} \end{cases}$$

Thus, $\oplus$ acts as a lexicographic maximum operator over path values with ties broken in an arbitrary but consistent way. In the language of closed semirings, it is the *summary operator*.

Define operator $\odot$ as $(v_1, \ell_1) \odot (v_2, \ell_2) \equiv (v_1 + \beta^{\ell_1} v_2, \ell_1 + \ell_2)$. The $\odot$ operator can be interpreted as a concatenation operator for a pair of paths. In the language of closed semirings, it is the *extension operator*.

Define $\bar{0} = (-\infty, -\infty)$ and $\bar{1} = (0, 0)$. Then $\bar{0}$ acts as a sort of path sink and $\bar{1}$ as the empty path. Finally, define

$$\begin{aligned} (v, \ell)^\star &= \bar{1} \oplus (v, \ell) \oplus ((v, \ell) \odot (v, \ell)) \oplus ((v, \ell) \odot (v, \ell) \odot (v, \ell)) \oplus \ldots \\ &= (0, 0) \text{ if } v \leq 0 \text{ and } (v\beta^\ell/(1 - \beta), +\infty) \text{ otherwise.} \end{aligned}$$

Here the star is being used as an operator, not a notational symbol as in V^*. It represents the maximum value of cycling around a path zero or more times.

Now, for $(\mathbb{S}, \oplus, \odot, \bar{0}, \bar{1})$ to be a closed semiring, a collection of properties must be satisfied.

1. $(\mathbb{S}, \oplus, \bar{0})$ is a monoid.

    - $\mathbb{S}$ is closed under $\oplus$. This is trivial since the result of $s_1 \oplus s_2$ is always either s_1 or s_2.
    - $\oplus$ is associative. This follows fairly easily from the definition because the lexicographic ordering is total and therefore independent of the order in which elements are combined.
    - $\bar{0}$ is an identity for $\oplus$. Since both components of $\bar{0}$ are $-\infty$, $\bar{0} \oplus s = s \oplus \bar{0} = s$ for all $s \in \mathbb{S}$.

2. $(\mathbb{S}, \odot, \bar{1})$ is a monoid.

- $\mathbb{S}$ is closed under $\odot$. This follows from the fact that $\mathbb{L}$ and $\mathbb{V}$ are closed under addition, $\mathbb{V}$ is closed under multiplication, and $\beta^{\ell} \in \mathbb{V}$ for $\ell \in \mathbb{L}$.
- $\odot$ is associative. This is easily verified algebraically.
- $\bar{1}$ is an identity for $\odot$. This is easy to see given that $\beta^0 = 1$ and $v \cdot 0 = 0$.

3. $\bar{0}$ is an annihilator: $(v_1, \ell_1) \odot (-\infty, -\infty) = (v_1 + \beta^{\ell_1} - \infty, \ell_1 - \infty) = \bar{0}$ and $(-\infty, -\infty) \odot (v_1, \ell_1) = (-\infty + \beta^{-\infty} v_1, -\infty + \ell_1) = \bar{0}$.

4. $\oplus$ is commutative. This follows from the commutativity of lexicographic maxima.

5. $\oplus$ is idempotent. Again, this follows from the idempotence of maxima.

6. $\odot$ distributes over $\oplus$. First, $s_1 \odot (s_2 \oplus s_3) = (s_1 \odot s_2) \oplus (s_1 \odot s_3)$ because the left-hand side is s_1 concatenated to whichever path has greater value, s_2 or s_1; and the right-hand side is the path with greater value between s_1 concatenated to s_2 or s_1 concatenated to s_3. Second, $(s_2 \oplus s_3) \odot s_1 = (s_2 \odot s_1) \oplus (s_1 \odot s_3)$ for similar reasons.

7. If $s_1, s_2, s_3, \ldots$ is a countable sequence of elements in $\mathbb{S}$ then the infinite summary $s_1 \oplus s_2 \oplus s_3 \oplus \cdots$ is well defined and in $\mathbb{S}$. This follows from the fact that $\mathbb{V}$ and $\mathbb{L}$ are closed under countable sequences of $\oplus$ operations. It is important that $+\infty \in \mathbb{V}$ for this reason.

8. Associativity, commutativity, and idempotence apply to infinite summaries, thus, any infinite summary can be rewritten as an infinite summary in which each term of the summary is included just once and the order of evaluation is arbitrary.

9. $\odot$ distributes over infinite summaries.

As a result, every deterministic MDP is a closed semiring.

Appendix C

Supplementary Information on Generalized MDPs

In this appendix, I prove important properties of a collection of summary operators, the contraction of dynamic-programming operators in the all-policies-proper case, the convergence of policy iteration, and the convergence of a doubly asynchronous stochastic process.

C.1 Summary Operators

In this section, I prove several properties associated with functions that summarize sets of values. These summary operators are important for defining generalized Markov decision processes, which involve summaries over the action set $\mathcal{U}$ and the set of next states $N(x, u)$ for each state-action pair (x, u).

Let I be a finite set and $h : I \rightarrow \mathbb{R}$. We define a *summary operator* $\bigodot$ over I to be a function that maps a real-valued function over I to a real number. The maximum operator $\max_{i \in I} h(i)$ and the minimum operator $\min_{i \in I} h(i)$ are important examples of summary operators.

Let h be a real-valued function over I. We say a summary operator $\bigodot$ is a *non-expansion* if it satisfies two properties:

$$\min_{i \in I} h(i) \leq \bigodot_{i \in I} h(i) \leq \max_{i \in I} h(i), \tag{C.1}$$

and

$$\left| \bigodot_{i \in I} h(i) - \bigodot_{i \in I} h'(i) \right| \leq \max_{i \in I} |h(i) - h'(i)|. \tag{C.2}$$

I will show that the max and min summary operators are both non-expansions, after proving a series of simpler results.

Let h and h be real-valued functions over I. For $i \in I$, let $\bigodot^i$ be the summary operator $\bigodot^i_{i' \in I} h(i') = h(i)$.

Theorem C.1 *The summary operator $\bigodot^i$ is a non-expansion.*

Proof: Condition C.1 requires that $\bigodot^i_{i' \in I} h(i') = h(i)$ lie between $\min_{i' \in I} h(i')$ and $\max_{i' \in I} h(i')$. This holds trivially.

To see that Condition C.2 holds, note that $|\bigodot^i_{i' \in I} h(i') - \bigodot^i_{i' \in I} h'(i')| = |h(i) - h'(i)| \leq \max_{i' \in I} |h(i') - h'(i')|$. $\square$

I next examine a more complicated set of non-expansions. For real-valued function h over I, let $\mathrm{ord}^n_{i \in I} h(i)$ be the nth largest value of $h(i)$ $(1 \leq n \leq |I|)$. According to this definition, $\mathrm{ord}^1_{i \in I} h(i) = \max_i h(i)$ and $\mathrm{ord}^{|I|}_{i \in I} h(i) = \min_i h(i)$. I will show that the ord^n summary operator is a non-expansion for all $1 \leq n \leq |I|$. To do this, I show that pairing the values of two functions in their sorted order minimizes the largest pairwise difference between the functions.

Lemma C.1 *Let h_1 and h_2 be real-valued functions over I and $i_1, i_2, i_3, i_4 \in I$. If $h_1(i_1) \leq h_1(i_2)$ and $h_2(i_3) \leq h_2(i_4)$, then*

$$\begin{aligned} & \max\{|h_1(i_1) - h_2(i_3)|, |h_1(i_2) - h_2(i_4)|\} \\ \leq \quad & \max\{|h_1(i_1) - h_2(i_4)|, |h_1(i_2) - h_2(i_3)|\}. \end{aligned}$$

Proof: Two bounds can be proven separately:

$$\begin{aligned} |h_1(i_1) - h_2(i_3)| &= \max\{h_1(i_1) - h_2(i_3), h_2(i_3) - h_1(i_1)\} \\ &\leq \max\{h_1(i_2) - h_2(i_3), h_2(i_4) - h_1(i_1)\} \\ &\leq \max\{|h_1(i_1) - h_2(i_4)|, |h_1(i_2) - h_2(i_3)|\}, \end{aligned}$$

and

$$\begin{aligned}
|h_1(i_2) - h_2(i_4)| &= \max\{h_1(i_2) - h_2(i_4), h_2(i_4) - h_1(i_2)\} \\
&\leq \max\{h_1(i_2) - h_2(i_3), h_2(i_4) - h_1(i_1)\} \\
&\leq \max\{|h_1(i_1) - h_2(i_4)|, |h_1(i_2) - h_2(i_3)|\}.
\end{aligned}$$

Combining these two inequalities proves the lemma. □

I use Lemma C.1 to create a bound involving the ord^n summary operator.

Lemma C.2 *Let h_1 and h_2 be real-valued functions over I. Then*

$$\max_n |\mathrm{ord}^n_{i\in I} h_1(i) - \mathrm{ord}^n_{i\in I} h_2(i)| \leq \max_{i \in I} |h_1(i) - h_2(i)|.$$

Proof: Both quantities in the inequality involve taking a maximum over differences between matched pairs of values. This lemma states that, of all possible matchings, pairing values with the same position in a sorted list of values gives the smallest maximum difference.

To prove this, I argue that, from any matching that violates the sorted order we can produce a matching that is "more sorted" without increasing the maximum difference (and perhaps decreasing it). The idea is that we can find a pair of pairs of values that are matched out of order, and swap the matching for that pair. By Lemma C.1, the resulting matching has a maximum difference no larger than the previous matching. After generating pairings that are more and more sorted, we eventually reach the totally sorted matching. Since the initial matching was arbitrary, the lemma follows. □

That ord^n is a non-expansion follows easily from Lemma C.2.

Theorem C.2 *The ord^n operator is a non-expansion for all $1 \leq n \leq |I|$.*

Proof: Condition C.1 is satisfied easily since it is always the case that $\mathrm{ord}^n_{i\in I} h(i) = h(i)$ for some $i \in I$.

To verify Condition C.2, let h_1 and h_2 be real-valued functions over I. It follows from Lemma C.2 that

$$\begin{aligned}
|\mathrm{ord}^n_{i\in I} h_1(i) - \mathrm{ord}^n_{i\in I} h_2(i)| &\leq \max_n |\mathrm{ord}^n_{i\in I} h_1(i) - \mathrm{ord}^n_{i\in I} h_2(i)| \\
&\leq \max_{i\in I} |h_1(i) - h_2(i)|.
\end{aligned}$$

Since n was arbitrary, the theorem is proved. $\square$

Theorems C.1 and C.2 state that two very specific classes of summary operators are non-expansions. The next theorem makes it possible to create more complex non-expansions by blending non-expansions together.

Theorem C.3 *If* $\bigodot^1$ *and* $\bigodot^2$ *are non-expansions, then for any* $0 \leq \nu \leq 1$*, the summary operator*

$$\bigodot_{i \in I}^{(1+2),\nu} h(i) = \nu \bigodot_{i \in I}^{1} h(i) + (1-\nu) \bigodot_{i \in I}^{2} h(i)$$

is a non-expansion.

Proof: Once again, Condition C.1 is not difficult to verify since the operators are being combined using a (convex) weighted average.

Condition C.2 follows from

$$\begin{aligned}
&\left| \bigodot_{i \in I}^{(1+2),\nu} h(i) - \bigodot_{i \in I}^{(1+2),\nu} h'(i) \right| \\
&= \left| \nu \bigodot_{i \in I}^{1} h(i) + (1-\nu) \bigodot_{i \in I}^{2} h(i) - \left(\nu \bigodot_{i \in I}^{1} h'(i) + (1-\nu) \bigodot_{i \in I}^{2} h'(i) \right) \right| \\
&= \left| \nu \left(\bigodot_{i \in I}^{1} h(i) - \bigodot_{i \in I}^{1} h'(i) \right) + (1-\nu) \left(\bigodot_{i \in I}^{2} h(i) - \bigodot_{i \in I}^{2} h'(i) \right) \right| \\
&\leq \nu \left| \bigodot_{i \in I}^{1} h(i) - \bigodot_{i \in I}^{1} h'(i) \right| + (1-\nu) \left| \bigodot_{i \in I}^{2} h(i) - \bigodot_{i \in I}^{2} h'(i) \right| \\
&\leq \nu \max_{i \in I} |h(i) - h'(i)| + (1-\nu) \max_{i \in I} |h(i) - h'(i)| = \max_{i \in I} |h(i) - h'(i)|.
\end{aligned}$$

The proof is easily extended to weighted averages of more than two operators. $\square$

The previous theorem demonstrated one way of making non-expansions out of other non-expansions by averaging. The next theorem shows a more sophisticated method for constructing non-expansions.

If $\bigodot^1$ is a summary operator over I_1, and $\bigodot^2$ is a summary operator over I_2, we define the *composition* of $\bigodot^1$ and $\bigodot^2$ to be a summary operator over $I_1 \times I_2$,

$$(\bigodot^1 \circ \bigodot^2)_{(i_1,i_2) \in I_1 \times I_2} h((i_1, i_2)) = \bigodot_{i_1 \in I_1}^{1} \bigodot_{i_2 \in I_2}^{2} h((i_1, i_2)).$$

Theorem C.4 *Let* $\bigodot = \bigodot^1 \circ \bigodot^2$ *for non-expansions* $\bigodot^1$ *over* I_1 *and* $\bigodot^2$ *over* I_2*. Then* $\bigodot$ *over* $I = I_1 \times I_2$ *is a non-expansion.*

Proof: Let h and h' be real-valued functions over I. For Condition C.1, we see that

$$\begin{aligned}
\bigodot_{(i_1,i_2)\in I} h((i_1,i_2)) &= \left(\bigodot^1 \circ \bigodot^2\right)_{(i_1,i_2)\in I} h((i_1,i_2)) \\
&= \bigodot_{i_1\in I_1}^1 \bigodot_{i_2\in I_2}^2 h((i_1,i_2)) \\
&\leq \max_{i_1\in I_1} \bigodot_{i_2\in I_2}^2 h((i_1,i_2)) \\
&\leq \max_{i_1\in I_1}\max_{i_2\in I_2} h((i_1,i_2)) \\
&\leq \max_{(i_1,i_2)\in I} h((i_1,i_2)).
\end{aligned}$$

The argument that $\bigodot_{(i_1,i_2)\in I} h((i_1,i_2)) \geq \min_{(i_1,i_2)\in I} h((i_1,i_2))$ is similar.

For Condition C.2,

$$\begin{aligned}
&\left|\bigodot_{(i_1,i_2)\in I} h((i_1,i_2)) - \bigodot_{(i_1,i_2)\in I} h'((i_1,i_2))\right| \\
&= \left|(\bigodot^1 \circ \bigodot^2)_{(i_1,i_2)\in I} h((i_1,i_2)) - (\bigodot^1 \circ \bigodot^2)_{(i_1,i_2)\in I} h'((i_1,i_2))\right| \\
&= \left|\bigodot_{i_1\in I_1}^1 \bigodot_{i_2\in I_2}^2 h((i_1,i_2)) - \bigodot_{i_1\in I_1}^1 \bigodot_{i_2\in I_2}^2 h'((i_1,i_2))\right| \\
&\leq \max_{i_1\in I_1}\left|\bigodot_{i_2\in I_2}^2 h((i_1,i_2)) - \bigodot_{i_2\in I_2}^2 h'((i_1,i_2))\right| \\
&\leq \max_{i_1\in I_1}\max_{i_2\in I_2} |h((i_1,i_2)) - h'((i_1,i_2))| = \max_{(i_1,i_2)\in I} |h((i_1,i_2)) - h'((i_1,i_2))|.
\end{aligned}$$

This proves that $\bigodot$ is a non-expansion. □

As a non-trivial application of the preceding theorems, I will show that the minimax summary operator, used in Markov games, is a non-expansion. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be finite sets. The *minimax* summary operator over $\mathcal{A}_1 \times \mathcal{A}_2$ is defined as

$$\text{minimax}_{(a_1,a_2)\in\mathcal{A}_1\times\mathcal{A}_2} h((a_1,a_2)) = \max_{\rho\in\Pi(\mathcal{A}_1)} \min_{a_2\in\mathcal{A}_2} \sum_{a_1\in\mathcal{A}_1} \rho[a_1] h((a_1,a_2)).$$

Let $\rho \in \Pi(\mathcal{A}_1)$ and let h_1 be a real-valued function over $\mathcal{A}_1$. Define

$$\bigodot_{a_1\in\mathcal{A}_1}^{\rho} h_1(a_1) = \sum_{a_1\in\mathcal{A}_1} \rho[a_1] h_1(a_1);$$

by Theorem C.3 and Theorem C.1, $\bigodot^\rho$ is a non-expansion. Let h be a real-valued function over $\mathcal{A}_1 \times \mathcal{A}_2$. By Theorem C.2, the minimum operator is a non-expansion.

Rewrite

$$\text{minimax}_{(a_1,a_2)\in\mathcal{A}_1\times\mathcal{A}_2} h((a_1,a_2)) = \max_{\rho\in\Pi(\mathcal{A}_1)} \left(\min \circ \bigodot^{\rho}\right)_{(a_2,a_1)\in\mathcal{A}_2\times\mathcal{A}_1} h((a_1,a_2));$$

minimax is a non-expansion by Theorem C.4 and the compactness of the set $\Pi(\mathcal{A}_1)$ of probability distributions over $\mathcal{A}_1$.

The class of non-expansions is quite broad. It is tempting to think that any operator that satisfies Condition C.1 will be a non-expansion. This is not the case.

Lemma C.3 *Define the boltzmann weighted average of h as*

$$\text{BOLTZ}^T_{i\in I} h(i) = \sum_{i\in I} h(i) \frac{e^{h(i)/T}}{\sum_{i\in I} e^{h(i)/T}}.$$

The operator BOLTZ^T *is not a non-expansion.*

Proof: Let $I = \{1,2\}$, $T = 1$, $h(1) = 100$, $h(2) = 1$, $h'(1) = 1$, and $h'(2) = 0$. For BOLTZ^T to be a non-expansion, it must be the case that Conditions C.1 and C.2 hold. Although Condition C.1 holds,

$$\begin{aligned}
&|\text{BOLTZ}^T_{i\in I} h(i) - \text{BOLTZ}^T_{i\in I} h'(i)| \\
= &\left| \sum_i h(i) \frac{e^{h(i)/T}}{\sum_i e^{h(i)/T}} - \sum_i h'(i) \frac{e^{h'(i)/T}}{\sum_i e^{h'(i)/T}} \right| \\
\approx &\ |(100+0) - (0.731+0)| = 99.269 > 99 = \max_{i\in I} |h(i) - h'(i)|,
\end{aligned}$$

proving that the operator is not a non-expansion. □

C.2 Contractions in the All-policies-proper Case

Consider a finite-state generalized Markov decision process[1] under the expected reward criterion satisfying the *all-policies-proper condition.* This condition states that one state, call it x_0, is a zero-reward absorbing state ($T(x_0,u,x_0) = 1$ and $R(x_0,u) = 0$ for all u), and every other state has a positive probability of reaching x_0 eventually, for any policy.

In this section, we work through a novel proof that, under these conditions, the dynamic-programming operator H is a contraction mapping under some weighted max

[1] Recall that generalized MDPs are defined to have a finite action set.

norm, even if $\beta = 1$. Alternate proofs for the MDP case are given by Bertsekas and Tsitsiklis [18], and Tseng [162].

Define $w(x_0) = 0$ and $w(x) = \max_u \left(1 + \sum_{x' \in N(x,u)} T(x,u,x') w(x')\right)$; that is, $w(x)$ is the maximum expected steps to absorption from state x over all policies. This is the Bellman equation for a simple all-policies-proper MDP, and is well defined [126]. Because of the all-policies-proper condition and the definition of w, $0 \leq w(x) < \infty$ for all $x \in \mathcal{X}$. Define $\beta_w = \beta \max_x((w(x) - 1)/w(x))$; it is strictly less than one because both $w(x)$ and $|\mathcal{X}|$ are finite.

Let V_1 and V_2 be value functions and Q_1 and Q_2 be Q functions with

$$Q_1(x,u) = R(x,u) + \beta \sum_{x' \in N(x,u)} T(x,u,x') V_1(x'),$$

and

$$Q_2(x,u) = R(x,u) + \beta \sum_{x' \in N(x,u)} T(x,u,x') V_2(x').$$

The definitions of H, w and β_w, along with the non-expansion properties of $\bigotimes$ imply

$$\begin{aligned}
\|HV_1 - HV_2\|_w &= \max_x \frac{|[HV_1](x) - [HV_2](x)|}{w(x)} \\
&= \max_x \frac{|[\bigotimes Q_1](x) - [\bigotimes Q_2](x)|}{w(x)} \\
&\leq \max_x \max_u \frac{|Q_1(x,u) - Q_2(x,u)|}{w(x)} \\
&\leq \max_x \max_u \beta \sum_{x' \in N(x,u)} T(x,u,x') \frac{|V_1(x') - V_2(x')|}{w(x)} \\
&\leq \max_x \max_u \beta \sum_{x' \in N(x,u)} T(x,u,x') \frac{w(x')}{w(x)} \frac{|V_1(x') - V_2(x')|}{w(x')} \\
&\leq \max_x \beta \max_u \sum_{x' \in N(x,u)} T(x,u,x') \frac{w(x')}{w(x)} \|V_1 - V_2\|_w \\
&\leq \max_x \beta \frac{w(x) - 1}{w(x)} \|V_1 - V_2\|_w \\
&\leq \beta_w \|V_1 - V_2\|_w.
\end{aligned}$$

This shows that H is a contraction mapping with respect to the weighting function w with contraction coefficient β_w.

Note that the weights w are can be determined by the solution of an MDP. By the reasoning in Theorem 2.1, each of the weights (and β_w) can be written using a polynomial number of bits. This is important for arguing that value iteration for all-policies-proper MDPs converges in pseudopolynomial time.

Let Q_1 and Q_2 be Q functions. Using reasoning similar to the above,

$$
\begin{aligned}
&\|KQ_1 - KQ_2\|_w \\
&= \max_x \max_u \frac{|[KQ_1](x,u) - [KQ_2](x,u)|}{w(x)} \\
&= \max_x \max_u \beta \sum_{x' \in N(x,u)} T(x,u,x') \frac{|[\bigotimes Q_1](x') - [\bigotimes Q_2](x')|}{w(x)} \\
&\leq \max_x \max_u \beta \sum_{x' \in N(x,u)} T(x,u,x') \max_{u'} \frac{|Q_1(x',u') - Q_2(x',u')|}{w(x)} \\
&\leq \max_x \beta \max_u \sum_{x' \in N(x,u)} T(x,u,x') \frac{w(x')}{w(x)} \max_{u'} \frac{|Q_1(x',u') - Q_2(x',u')|}{w(x')} \\
&\leq \max_x \beta \max_u \sum_{x' \in N(x,u)} T(x,u,x') \frac{w(x')}{w(x)} \max_{x'} \max_{u'} \frac{|Q_1(x',u') - Q_2(x',u')|}{w(x')} \\
&\leq \max_x \beta \frac{w(x)-1}{w(x)} \|Q_1 - Q_2\|_w \\
&\leq \beta_w \|Q_1 - Q_2\|_w,
\end{aligned}
$$

demonstrating that the K operator on Q functions is also a contraction mapping with respect to the w weighted max norm.

It is interesting to ask whether this result holds true for different definitions of $\bigoplus$, such as the minimization operator. In fact, the result does not hold for either minimization or maximization, unless the state space is entirely *acyclic* (no policy has a positive probability path from a state to back itself).

C.3 Monotonicity of Several Operators

Section C.1 describes a collection of important non-expansion operators based on element selection, ordering, convex combinations, and composition. All of these operators obey an additional monotonicity property as well.

Summary operator $\bigodot$ is monotonic if, for all real-valued functions h and h' over a finite set I, $h(i) \geq h'(i)$ for all $i \in I$ implies

$$\bigodot_{i \in I} h(i) \geq \bigodot_{i \in I} h'(i).$$

Theorem C.5 *The following summary operators are monotonic:* $\bigodot^i$ *for all* $i \in I$*,* ord^n *for all* $1 \leq n \leq |I|$*,* $\bigodot^{(1+2),\nu}$ *for all* $0 \leq \nu \leq 1$ *if* $\bigodot^1$ *and* $\bigodot^2$ *are monotonic, and* $\bigodot^1 \circ \bigodot^2$ *if* $\bigodot^1$ *and* $\bigodot^2$ *are monotonic.*

Proof: The monotonicity of $\bigodot^i$, $\bigodot^{(1+2),\nu}$, and $\bigodot^1 \circ \bigodot^2$ follow immediately from their definitions. The monotonicity of ord^n can be proven by considering the effect of increasing $h(i)$ to $h'(i)$ for each $i \in I$, one at a time. A simple case analysis shows that each increase in $h(i)$ cannot decrease the value of $\mathrm{ord}^n_{i \in I} h(i)$. □

C.4 Policy-Iteration Convergence Proof

In this section, I develop the necessary results to show that the generalized policy-iteration algorithm of Section 3.3.3 converges to the optimal value function. I will first prove several simple lemmas that illuminate the fundamental properties of value functions in maximizing generalized MDPs.

First, for maximizing generalized MDPs, a single step of value iteration on a value function associated with a mapping ω, results in a value function that is no smaller.

Lemma C.4 *For all* $\omega : \mathcal{X} \rightarrow \mathcal{R}$*,* $HV^\omega \geq V^\omega$*.*

Proof: From Equation 3.4, the constraints on $\bigotimes$, and the definition of V^ω,

$$\begin{aligned}
[HV^\omega](x) &= \bigotimes_u^{(x)} \left(R(x,u) + \beta \bigoplus_{x'}^{(x,u)} V^\omega(x') \right) \\
&= \max_{\rho \in \mathcal{R}} \bigotimes_u^{\rho,(x)} \left(R(x,u) + \beta \bigoplus_{x'}^{(x,u)} V^\omega(x') \right) \\
&\geq \bigotimes_u^{\omega(x),(x)} \left(R(x,u) + \beta \bigoplus_{x'}^{(x,u)} V^\omega(x') \right) = V^\omega(x).
\end{aligned}$$

□

Let H^ω be the dynamic-programming operator associated with the mapping ω

$$[H^\omega V](x) = \bigotimes_u^{\omega(x),(x)} \left(R(x,u) + \beta \bigoplus_{x'}^{(x,u)} V(x') \right).$$

The next lemma says that the monontonicy properties of $\bigotimes$ and $\bigoplus$ carry over to H and H^ω.

Lemma C.5 *The mappings H and H^ω are monotonic for maximizing generalized* MDP*s.*

Proof: For value functions V and V', we want to show that if $V \geq V'$, then $HV \geq HV'$ and $H^\omega V \geq H^\omega V'$. This follows easily from the definitions and the monotonicity of the operators involved. □

Theorem 3.5 states that the optimal value function dominates the value functions for all ω. I will now prove this using Lemmas C.4 and C.5.

From Lemma C.4, we have that $V^\omega \leq HV^\omega$ for all ω. Combining this with the result of Lemma C.5, we have $HV^\omega \leq H(HV^\omega)$. By induction and transitivity, $V^\omega \leq (H)^k V^\omega$ for all integers $k \geq 0$ where $(H)^k$ corresponds to the application of H repeated k times. Because $\lim_{k\to\infty}(H)^k V^\omega = V^*$, it is not difficult to show that $V^\omega \leq V^*$, proving Theorem 3.5.

The final result we need relates the convergence of policy iteration to that of value iteration. Let U_t be the iterates of value iteration and V_t be the iterates of policy iteration, starting from the same initial value function. Let $\omega_t : \mathcal{X} \to \mathcal{R}$ be the sequence of mappings such that $V_t = V^{\omega_t}$.

Lemma 3.6 states that, for all t and $x \in \mathcal{X}$, $U_t(x) \leq V_t(x) \leq V^*(x)$. We proceed by induction. Clearly $U_0(x) \leq V_0(x)$, because they are defined to be equal. Now, assume that $U_t(x) \leq V_t(x) \leq V^*(x)$. By Lemma C.5, $HU_t(x) \leq HV_t(x)$. By definition, $HU_t(x) = U_{t+1}(x)$, and by an argument similar to the proof of Theorem 3.5,

$$HV_t = H^{\omega_{t+1}} V_t \leq (H^{\omega_{t+1}})^k V_t \leq V^{\omega_{t+1}} = V_{t+1}.$$

Therefore, $U_{t+1}(x) \leq V_{t+1}(x)$. By Theorem 3.5, $V_{t+1}(x) = V^{\omega_{t+1}} \leq V^*(x)$, completing the proof of Lemma 3.6.

Lemma 3.6 and Lemma 3.4 together imply the convergence of policy iteration. Lemma 3.6 also provides a bound on the convergence rate of the algorithm; it is no slower than value iteration, but perhaps faster.

C.5 A Stochastic-Convergence Proof

In this section, I prove Theorem 3.7, which is useful for proving the convergence of reinforcement-learning algorithms in generalized MDPs.

The proof relies critically on the following lemma concerning the convergence of stochastic processes.

Lemma C.6 *Let $\mathcal{Z}$ be an arbitrary set and consider the sequence*

$$x_{t+1}(z) = g_t(z)x_t(z) + f_t(z)\|x_t(z) + \epsilon_t(z)\|,$$

where $z \in \mathcal{Z}$ and $\epsilon_t(z) \geq 0$ converges to zero. Assume that for all k,

$$\lim_{n\to\infty} \Pi_{t=k}^{n} g_t(z) = 0$$

uniformly in z with probability 1 and $f_t(z) \leq \beta(1-g_t(z))$ with probability 1. Then $x_t(z)$ converges to 0 with probability 1.

Proof: The proof is in a paper by Szepesvári and Littman [158]. A similar claim is proven by Jaakkola, Jordan and Singh [69]. □

Let H be a contraction mapping with respect to a weighted max norm with fixed point V^*, and let H_t approximate H at V^*. Let V_0 be an arbitrary value function, and define $V_{t+1} = H_t(V_t, V_t)$. If there exist functions $0 \leq F_t(x) \leq 1$ and $0 \leq G_t(x) \leq 1$ satisfying the conditions below with probability one, then V_t converges uniformly to V^* with probability 1:

1. for all value functions U_1 and U_2 and all $x \in \mathcal{X}$,

$$|(H_t(U_1, V^*))(x) - (H_t(U_2, V^*))(x)| \leq G_t(x)\|U_1 - U_2\|;$$

2. for all value functions U and V, and all $x \in \mathcal{X}$,

$$|(H_t(U, V^*))(x) - (H_t(U, V))(x)| \leq F_t(x)\|V^* - V\|;$$

3. for all $k > 0$, $\Pi_{t=k}^{n} G_t(x)$ converges to zero uniformly in x as n increases; and,

4. there exists $0 \leq \beta < 1$ such that for all $x \in \mathcal{X}$ and large enough t,

$$F_t(x) \leq \beta(1 - G_t(x)).$$

To prove this, we will define a sequence of auxillary functions, U_t, that is guaranteed to converge, and relate the convergence of V_t to the convergence of U_t. Let U_0 be an arbitrary value function and let $U_{t+1} = H_t(U_t, V^*)$. Since H_t approximates H, U_t converges to $HV^* = V^*$ with probability 1 uniformly over $\mathcal{X}$. We will show that

$\|U_t - V_t\|$ converges to zero with probability 1, which implies that V_t converges to V^*. Let

$$\delta_t(x) = |U_t(x) - V_t(x)|$$

and let

$$\Delta_t(x) = |U_t(x) - V^*(x)|.$$

We know that $\Delta_t(x)$ converges to zero because U_t converges to V^*.

By the triangle inequality and the constraints on H_t, we have

$$\begin{aligned}
\delta_{t+1}(x) &= |H_t(U_t, V^*)(x) - H_t(V_t, V_t)(x)| \\
&\leq |H_t(U_t, V^*)(x) - H_t(V_t, V^*)(x)| + |H_t(V_t, V^*)(x) - H_t(V_t, V_t)(x)| \\
&\leq G_t(x)\|U_t - V_t\| + F_t(x)\|V^* - V_t\| \\
&\leq G_t(x)\delta_t(x) + F_t(x)\|V^* - V_t\| \\
&\leq G_t(x)\delta_t(x) + F_t(x)(\|V^* - U_t\| + \|U_t - V_t\|) \\
&\leq G_t(x)\delta_t(x) + F_t(x)(\|\delta_t(x)\| + \Delta_t(x))
\end{aligned}$$

Inequality C.3 is now in the correct form for Lemma C.6, which tells us that $\delta_t(x)$ goes to zero with probability 1. This proves Theorem 3.7.

Appendix D

Supplementary Information on Alternating Markov Games

In this appendix, I prove that Markov games in which players switch turns after every action and Markov games in which players switch turns according to the current state are equivalent in complexity.

D.1 Equivalence to Strictly Alternating Markov Games

There are two models that might properly be called "alternating Markov games"; in one, control of the play strictly alternates between the two players, in the other, control remains with a player for an unspecified number of actions before switching to the other player. The two models are equivalent in the sense that a polynomial-time algorithm for one could be used to solve instances of the other model in polynomial time.

A strictly alternating Markov game (SAMG) is defined by the tuple

$$\langle \mathcal{S}, \mathcal{A}_1, \mathcal{A}_2, T, R, \beta \rangle$$

and play strictly alternates between the two players. An alternating Markov game (AMG) is defined by $\langle \mathcal{S}_1, \mathcal{S}_2, \mathcal{A}_1, \mathcal{A}_2, T, R, \beta \rangle$, and control belongs to player 1 if the current state is in $\mathcal{S}_1$, and it belongs to player 2 otherwise. Given a SAMG, we can create an equivalent AMG by duplicating the state space into sets $\mathcal{S}_1$ and $\mathcal{S}_2$ so that the state encodes the turn of the player, and redefining transitions so they alternate between the two copies of the states.

Given an AMG, we can also create an equivalent SAMG. The difficulty is that the first player takes an unspecified number of actions in the AMG before control is turned over to the second player. How can we make control alternate on every turn? We can do this by introducing a number of "dummy" states where one or the other player nominally has control, but from which the state transition is actually completely determined.

In more detail, consider an AMG $\mathcal{G} = \langle \mathcal{S}_1, \mathcal{S}_2, \mathcal{A}_1, \mathcal{A}_2, T, R, \beta \rangle$. We will define a SAMG $\mathcal{G}' = \langle \mathcal{S}, \mathcal{A}_1, \mathcal{A}_2, T', R', \beta' \rangle$ such that the solution to $\mathcal{G}'$ can be used to quickly find the solution to $\mathcal{G}$. The set $\mathcal{S}$ consists of all the states in $\mathcal{S}_1$ and $\mathcal{S}_2$ as well as the dummy states. For every state $s_1 \in \mathcal{S}_1$ and action $a_1 \in \mathcal{A}_1$ we introduce a new state s_2 into $\mathcal{S}$ and define the transition function T' so that action a_1 from state s_1 results in a deterministic transition to state s_2, and from s_2, any action $a_2 \in \mathcal{A}_2$ results in the same state transitions defined by T for state-action pair (s_1, a_1). In this way, state s_2 "intercepts" the transition, resulting in a (dummy) action for player 2 after player 1's action. This ensures that every action for player 1 is immediately followed by an action for player 2.

We need to also introduce dummy states that intercept the incoming transitions to the states in $\mathcal{S}_2$. This requires one dummy state added to $\mathcal{S}$ for each state in $\mathcal{S}_2$. Once this transformation is complete, each transition in $\mathcal{G}$ has been replaced by a pair of transitions in $\mathcal{G}'$: a transition for a $\mathcal{S}_1$ state followed by a dummy move for player 2, *or* a dummy move for player 1 followed by a transition for an $\mathcal{S}_2$ state. The transformation has the critical property that the probability of reaching some state s in t steps in $\mathcal{G}$ under some policy is exactly equal to the probability of reaching s in $2t \pm 1$ steps under the analogous policy in $\mathcal{G}'$.

It remains to be shown how to modify R and β to ensure that the optimal value of state s in $\mathcal{G}$ is equal to its optimal value in $\mathcal{G}'$. To a first approximation, this is quite easy. Because one step in $\mathcal{G}$ is equivalent to two steps in $\mathcal{G}'$, we need the discount factor to decay half as fast: $\beta' = \sqrt{\beta}$. We then modify the rewards so that all the dummy states have only zero-reward transitions, the states in $\mathcal{S}_1$ have the same rewards in $\mathcal{G}'$ that they have in $\mathcal{G}$, and the states in $\mathcal{S}_2$ have their rewards increased by a factor of $1/\beta'$. It is not hard to show that the optimal value of a state $s \in \mathcal{S}_1$ in $\mathcal{G}$ is precisely equal to the optimal value of the analogous state in $\mathcal{G}'$ and that the optimal value of a state $s \in \mathcal{S}_2$ in $\mathcal{G}$ is precisely equal to the optimal value of the analogous state in $\mathcal{G}'$ multiplied by β'.

A major difficulty remains. Even if $\mathcal{G}$ is specified using only rational numbers of B or fewer bits, the discount factor β' for $\mathcal{G}'$ may be irrational. I now argue that there is a rational value for β' that is close enough to $\sqrt{\beta}$ to ensure an optimal policy for $\mathcal{G}'$ is optimal for $\mathcal{G}$ and yet can be specified with a number of bits polynomial in the size of $\mathcal{G}$ and B.

The argument has three parts. First, there is a value $\epsilon > 0$ such that an ϵ-optimal value function yields an optimal policy. This follows from the argument in the proof of Lemma 2.1.

Second, there is a value $\delta > 0$ such that using a discount factor of $\beta + \delta$ in place of β when evaluating a policy results in a value function that is no more than ϵ away from the true value function for that policy. Finally, the first two parts together imply that we can use a polynomial-bit approximation of $\sqrt{\beta}$ in the construction described earlier and still be able to identify optimal policies.

We choose $\delta \leq \epsilon(1-\beta)^2/(M+(1-\beta)\epsilon)$. Let V^π be the value function for some policy π under discount factor β, and let $V^{\pi'}$ be the value function for π under discount factor $\beta+\delta$. Let s^* be the state for which $|V^{\pi'}(s^*) - V^\pi(s^*)|$ is maximized. Once again, let M be the magnitude of the largest absolute immediate reward. Substituting definitions reveals

$$\begin{aligned}
&|V^{\pi'}(s^*) - V^\pi(s^*)| \\
&\quad = \ |\beta \sum_{s'} T(s,\pi(s),s')(V^{\pi'}(s') - V^\pi(s')) + \delta \sum_{s'} T(s,\pi(s),s') V^{\pi'}(s')| \\
&\quad \leq \ \beta \sum_{s'} T(s,\pi(s),s') |V^{\pi'}(s') - V^\pi(s')| + \delta \sum_{s'} T(s,\pi(s),s') |V^{\pi'}(s')| \\
&\quad \leq \ \beta |V^{\pi'}(s^*) - V^\pi(s^*)| + \delta M/(1-(\beta+\delta))
\end{aligned}$$

Solving for $|V^{\pi'}(s^*) - V^\pi(s^*)|$ and noting that

$$\begin{aligned}
1 - \beta - \delta &= 1 - \beta - \epsilon(1-\beta)^2/(M + (1-\beta)\epsilon) \\
&= ((M + (1-\beta)\epsilon)(1-\beta) - \epsilon(1-\beta)^2)/(M + (1-\beta)\epsilon) \\
&= (M(1-\beta))/(M + (1-\beta)\epsilon)
\end{aligned}$$

lets us continue with

$$
\begin{aligned}
|V^{\pi'}(s^*) - V^{\pi}(s^*)| &\leq \delta M/(1-(\beta+\delta))/(1-\beta) \\
&\leq \frac{\delta M}{(1-\beta)(1-(\beta+\delta))} \\
&\leq \frac{\epsilon(1-\beta)^2 M}{(1-\beta)(1-(\beta+\delta))(M+(1-\beta)\epsilon)} \\
&\leq \frac{\epsilon(1-\beta)^2 M}{(1-\beta)(1-(\beta+\delta))(M+(1-\beta)\epsilon)} \\
&\leq \frac{\epsilon(1-\beta)^2 M(M+(1-\beta)\epsilon)}{(1-\beta)(M+(1-\beta)\epsilon)(M(1-\beta))} \\
&\leq \epsilon,
\end{aligned}
$$

as desired.

From the expressions for δ and ϵ, it is not hard to show that the number of bits of accuracy needed in the computation of $\sqrt{\beta}$ is polynomial in the necessary parameters. Therefore, an AMG can be turned into an equivalent SAMG with only a polynomial increase in size. If the resulting SAMG can be solved in polynomial time, so can the original AMG.

Appendix E

Supplementary Information on Markov Games

In this appendix, I present a new result concerning the optimal value function of a deterministic Markov game.

E.1 A Deterministic Markov Game with an Irrational Value Function

The existence of deterministic stationary optimal policies for MDPs and alternating Markov games makes it easy to show that these models can be solved in finite time, simply by enumerating the possible policies and evaluating each one by solving a system of linear equations.

Markov games are different, in that optimal policies are sometimes stochastic. Of course, if the probabilities needed to express an optimal policy can be written as rational numbers (consider the "Rock, Paper, Scissors" example from Chapter 5), it still might be possible to identify an optimal policy in finite time. This is not generally the case for Markov games, however; even if the transitions, rewards, and discount factor are all represented by rational numbers, the optimal value of the game and the optimal stochastic policy can both require irrational numbers to express.

This fact was mentioned in Shapley's original paper [143], and a specific example appears in Vrieze's survey article [170]. Vrieze's example is a stochastic Markov game, and, because deterministic models are often easier to solve, there is reason to believe

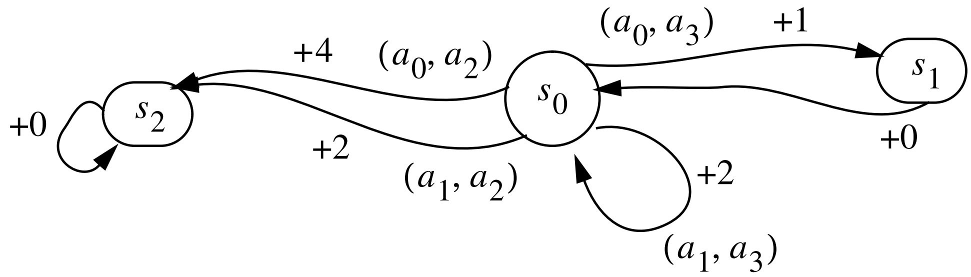

Figure E.1: A deterministic Markov game with rational rewards and $\beta = 3/4$ for which the optimal value function is irrational.

$$\begin{array}{lcllcl} \pi_1(s_0, a_0) & = & 1/2 & (-5+\sqrt{41}) & \approx & .702 \\ \pi_1(s_0, a_1) & = & 1/2 & (7-\sqrt{41}) & \approx & .298 \\ \pi_2(s_0, a_2) & = & 1/12 & (-1+\sqrt{41}) & \approx & .450 \\ \pi_2(s_0, a_3) & = & 1/12 & (13-\sqrt{41}) & \approx & .550 \end{array}$$

Table E.1: The optimal pair of stochastic policies for a deterministic Markov game.

that the deterministic case might be simpler. This does not appear to be the case for Markov games—deterministic Markov games can have irrational value functions even if all rewards and the discount factor are specified using rational numbers.

Consider the three-state deterministic Markov game illustrated in Figure E.1. In this game, state s_2 is a zero-reward absorbing state, state s_1 is a zero-reward state with deterministic transitions to state s_0, and state s_0 results in a transition to s_0, s_1, or s_2, depending on the actions chosen by the two players. From state s_0, player 1 must choose between actions a_0 and a_1 and player 2 must choose between actions a_2 and a_3; in the figure, transitions from state s_0 are labeled by pairs of actions.

The value of state s_0 in this game is $V^*(s_0) = -3 + \sqrt{31} \approx 3.403$, which was found by solving a quadratic equation. This can be verified by evaluating the stochastic policies in Table E.1. These policies are optimal because, if the agent adopts π_1, it guarantees itself a value of $V^*(s_0)$ regardless of the opponent's policy. Similarly, if the opponent adopts π_2, it guarantees itself $V^*(s_0)$ regardless of the opponent's policy.

The fact that optimal value functions can be irrational does not directly rule out the possibility that an exact algorithm exists; for example, if the values are all solutions to quadratic equations, then perhaps the equations themselves can be used to represent the values. However, no radix-type representation for the numbers will result in an exact algorithm.

Appendix F

Supplementary Results on POMDPs

In this appendix, I show that solving deterministic POMDPs is PSPACE-hard, and solving stochastic POMDPs is EXPTIME-hard. I also describe an example, due to Chrisman, of a difficult POMDP for Q-learning.

F.1 Hardness of Deterministic POMDPs

In this section, I prove that the problem of determining whether a deterministic POMDP has an infinite-horizon policy with zero reward is PSPACE-hard. The proof does not make use of the observation set, therefore the hardness result applies to unobservable POMDPs as well.

The proof relates deterministic unobservable POMDPs to finite-state automata. A finite-state automaton is much like a deterministic MDP with a finite set of states $\mathcal{S}$ and actions $\mathcal{A}$, a next-state function N, and an initial state s_0. A subset F of the states of the automaton are called *accepting states*. A finite-state automaton accepts string $\ell \in \mathcal{A}^\star$ (the star superscript denotes the Kleene star) if the state s reached after executing the string of actions ℓ starting from state s_0 is an accepting state (i.e., it is in F).

Lemma F.1 *Given a finite-state automaton $\mathcal{F} = \langle \mathcal{S}, \mathcal{A}, N, s_0, F\rangle$, there is a deterministic, unobservable, boolean-reward* POMDP *problem $\mathcal{M} = \langle \mathcal{S} \cup \{accept\}, \mathcal{A} \cup \{accept\}, N',$ $R, x_0\rangle$ such that $\mathcal{F}$ accepts string $\ell \in \mathcal{A}^\star$ if and only if executing the action sequence ℓ*

followed by "accept" results in zero reward and a transition to a zero-reward absorbing state in $\mathcal{M}$.

Proof: Create $\mathcal{M}$ by making a copy of $\mathcal{F}$ so that every transition in $\mathcal{F}$ is a zero-reward transition in $\mathcal{M}$. Define the state *accept* to be a zero-reward absorbing state. For every state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, define $N'(s,a) = N(s,a)$. For every accepting state $s_f \in F$, define $N'(s_f, \mathit{accept}) = \mathit{accept}$ and $R(s_f, \mathit{accept}) = 0$, and for every other state s in $\mathcal{S}$, make $N'(s, \mathit{accept}) = s$ and $R(s, \mathit{accept}) = -1$. Define $x_0[s_0] = 1$ and $x_0[s'] = 0$ for all $s' \neq s$.

The resulting POMDP satisfies the requirements of the lemma. □

Given a finite collection of finite-state automata $\mathcal{F}_1, \ldots, \mathcal{F}_k$, all with the same action set $\mathcal{A}$ but disjoint state sets $\mathcal{S}_1, \ldots \mathcal{S}_k$, we say that string $\ell \in \mathcal{A}^\star$ is in the *intersection* of the sets of strings accepted by the finite-state automata if ℓ is accepted by all k automata.

We can create a deterministic, unobservable, boolean-reward POMDP problem that is equivalent to a collection of finite-state automata.

Lemma F.2 *Given a collection of k finite-state automata, $\mathcal{F}_i = \langle \mathcal{S}_i, \mathcal{A}, N_i, s_{i,0}, F_i \rangle$ for each $1 \leq i \leq k$, there is a deterministic, unobservable, boolean-reward* POMDP *problem $\mathcal{M} = \langle \bigcup_i \mathcal{S}_i \cup \{\mathit{accept}\}, \mathcal{A} \cup \{\mathit{accept}\}, N', R, x_0 \rangle$ such that string $\ell \in \mathcal{A}^\star$ is in the intersection of the strings accepted by the finite-state automata if and only if executing the action sequence ℓ followed by "accept" results in zero reward and a transition to a zero-reward absorbing state in $\mathcal{M}$.*

Proof: The rewards R and the next state function N' are defined analogously to their definitions in Lemma F.1. We define $x_0[s_{i,0}] = 1/k$ for each $1 \leq i \leq k$ and zero otherwise. Thus, in the initial state, there is a probability of $1/k$ of being in each initial state of the collection of finite-state automata.

If, upon presentation of an action sequence ℓ, finite-state automaton i is in state s_i for each i, then, in the information state x for $\mathcal{M}$, $x[s_i] = 1/k$ for each i. As a result, the POMDP simulates all k of the finite-state automata concurrently. An *accept* action has zero reward if only if all k machines are in accepting states. □

The *finite-state-automata-intersection problem* is defined by a collection of finite-state automata. The problem is to decide whether there is any sequence $\ell \in \mathcal{A}^\star$ in the intersection of the sets of strings accepted by the finite-state automata; we call such a string ℓ an accepting string. The POMDP construction from Lemma F.2 cannot be used directly to determine the existence of an accepting string, because any policy that never issues the *accept* action will have the same value as one that issues *accept* after an accepting string. To make sure that policies are penalized for not issuing an accepting string when one exists, we can make use of a standard result that states that if there is an accepting string, then there must be an accepting string no more than $\prod_i |\mathcal{S}_i|$ symbols long. By adding a counter to the POMDP construction from Lemma F.2, we can use it solve the finite-state-automata-intersection problem.

Lemma F.3 *Given a collection of k finite-state automata $\mathcal{F}_i = \langle \mathcal{S}_i, \mathcal{A}, N_i, s_{i,0}, F_i \rangle$, for each $1 \leq i \leq k$, there is a deterministic, unobservable, boolean-reward* POMDP *problem $\mathcal{M} = \langle \mathcal{S}_i \cup \{accept\} \cup \{s_{i,j} | 1 \leq i \leq k, 1 \leq j \leq |\mathcal{S}_i|\} \cup \{inc, act\}, \mathcal{A} \cup \{accept\} \cup \{inc_i | 1 \leq i \leq k\}, N', R, x_0 \rangle$ such that the intersection of the strings accepted by the finite-state automata is non-empty if and only if there is a zero-reward policy for $\mathcal{M}$.*

Proof: The definitions from Lemma F.2 all apply here. One slight difference is that the initial state has $x_0[s_{i,0}] = 1/(2k+1)$ instead of $1/k$. The "new" states $s_{i,j}$, *inc*, and *act*, and actions inc_i will be used to implement a set of counters that will issue a negative reward if no *accept* action is chosen over the course of a sequence of $\prod_i |\mathcal{S}_i|$ actions.

The initial probability on the *inc* and *act* states is $x_0[inc] = 0$ and $x_0[act] = 1/(2k+1)$. The inc_i actions result in a transition from *inc* to *act*, and the other actions $a \in \mathcal{A}$ result in a transition from *act* to *inc*. If any action $a \in \mathcal{A}$ is selected from the *inc* state, there is a reward of -1. This forces the optimal policy to alternate between actions in $\mathcal{A}$ and inc_i actions.

As before, the actions in $\mathcal{A}$ correspond to transitions in all the finite-state automata simultaneously. The inc_i actions are used to implement a counter using the $s_{i,j}$ states. In the initial distribution, $x_0[s_{i,1}] = 1/(2k+1)$ for all i, and $x_0[s_{i,j}] = 0$ for all i and $j \geq 2$. This represents the reset state for the counters.

There is one counter for each finite-state automaton in the collection; counter 1 is the low-order counter, and counter k is the high-order counter. Action inc_i increments counter i, resets all the lower order counters, and does not change any of the higher order counters,

$$N(s_{i',j}, inc_i) = \begin{cases} s_{i',0}, & i' < i, \\ s_{i,j+1}, & j \leq |\mathcal{S}_i| - 1, i' = i, \\ s_{i',j}, & i' > i. \end{cases}$$

The reward for all these transitions is zero. Counter i is full when the probability of being in state $s_{i,|\mathcal{S}_i|}$ is non-zero. Incrementing a full counter results in negative reward, $R(s_{i,|\mathcal{S}_i|}, inc_i) = -1$.

To maximize the number of inc_i actions before a negative reward, a policy would issue inc_1 until the low-order counter was full, then inc_2 to increment the second counter and reset the first counter, then inc_1 once again. After $\prod_i |\mathcal{S}_i|$ increments of this kind, all k counters will be full. Any inc_i action at this point results in a negative reward.

If there is an accepting string for all the automata, a zero-reward policy would alternate between selecting actions corresponding to the shortest possible accepting string (any string shorter than $\prod_i |\mathcal{S}_i|$ would do), and the appropriate inc_i actions. Then, once the automata are all in their accepting states, issuing the *accept* action ensures the policy a total reward of zero.

If there is no accepting string, it is impossible to avoid a negative reward; if the *accept* action is selected, this will result in a negative reward, and if more than $2\prod_i |\mathcal{S}_i|$ steps elapse without issuing *accept*, a negative reward will be received.

Thus, the resulting POMDP problem has a zero-reward policy if and only if there is a string in the intersection of the sets of strings accepted by the given finite-state automata. □

Theorem F.1 *The deterministic, unobservable, boolean-reward* POMDP *problem is P-SPACE-hard.*

Proof: This theorem follows easily from the reduction in Lemma F.3 and the PSPACE-completeness of the finite-state-automata-intersection problem [55]. □

F.2 Hardness of Stochastic POMDPs

In this section, I show that solving infinite-horizon boolean-reward POMDPs with stochastic transitions and observations is EXPTIME-hard by showing that solving such POMDPs yields a solution to a particular type of game on boolean formulas.

The game was devised by Stockmeyer and Chandra [151] in their paper linking combinatorial two-player games to the class EXPTIME. The specific EXPTIME-hard game I use in this section is referred to as G_4, or "Peek," in their paper. It is a particular kind of deterministic alternating Markov game played by two players taking turns changing values of boolean variables, in an attempt to make a given formula evaluate to "true."

The game is defined by a choice of which player moves first, disjoint sets X and Y of variables, an initial assignment for these variables, and a disjunctive-normal-form boolean formula defined over the variables with 13 literals (variables or negations of variables) per term. Such a formula is called a "13-DNF" formula. The two players take turns changing the value of at most one of the variables in the formula; player 1 can only change the value of variables in X, and player 2 can only change the value of variables in Y. The game is over when the 13-DNF formula evaluates to true with the winner being the player whose action caused this to happen. The decision problem is whether there is a winning strategy for player 2 from the initial assignment.

I will show that every game of this form has an equivalent boolean-reward POMDP such that player 2 has a winning strategy in the game if and only if the optimal policy in the POMDP from a given initial state has negative expected reward. I use the set of states of the POMDP to represent the variable assignments and actions in the game, where the states with non-zero probability in the POMDP encode the state of the game.

In the POMDP, the agent plays the role of player 1. Of course, in a POMDP there is no second player; however, we can use the stochastic transitions to represent the actions of the second player. Because we judge the optimal policy by whether or not it achieves zero reward, any probability of encountering a negative reward amounts to certain failure. This makes it possible to assume that the worst possible transition occurs each time there is a choice; stochastic transitions are equivalent to worst-case transitions.

A game instance is a tuple $\mathcal{G} = \langle \tau, X, Y, F, \alpha \rangle$ where $\tau \in \{1, 2\}$ is the first player to move, X is the set of variables that player 1 can change, Y is the set of variables that player 2 can change, F is the 13-DNF formula for deciding termination represented as a set of terms each of which is a set of 13 literals, and $\alpha : X \cup Y \rightarrow \{\text{true}, \text{false}\}$ is an initial assignment of the variables. Given $\mathcal{G}$, we define the equivalent boolean-reward POMDP problem $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{Z}, O, S_0 \rangle$ as described below.

The set $\mathcal{S}$ of states consists of the literals, $X \cup \{\bar{x} | x \in X\} \cup Y \cup \{\bar{y} | y \in Y\}$ (the "literal-related" states); the state X; states for each possible action of player 2, $\{pass\} \cup \{tog(y) | y \in Y\} \cup \{togwin(y,t) | y \in Y, t \in F\}$ (the "player-2-action" states); and states "start" and "done." Only "start" is in the set of non-zero probability initial states, S_0.

The set $\mathcal{A}$ of actions consists of an action "start" for starting the game; actions for toggling the boolean value of each of the variables, $\{tog(x) | x \in X\} \cup \{tog(y) | y \in Y\}$; an action *pass* for passing; actions for simultaneously flipping a variable and declaring a win for player 1 by satisfying a particular term in the formula F, $\{togwin(x,t) | x \in X, t \in F\}$; and actions for challenging a win declared by player 2, $\{challenge(y,t,l) | y \in Y, t \in F, l \in L\}$, where L is the set of literals.

It is via the observations that player 2's choices of actions are realized. The set $\mathcal{Z}$ of observations consists of an element for each of player 2's actions, $pass \cup \{tog(y) | y \in Y\} \cup \{togwin(y,t) | y \in Y, t \in F\}$.

Roughly, here is how the states of the POMDP represent the state of the game. The information state of the POMDP is captured by the set of states which have non-zero probability. An assignment to the variables of the formula is represented by non-zero probabilities on the literal-related states; in particular, if a variable v is true, state v has non-zero probability while state $\bar{v}$ has zero probability; if a variable v is false, state v has zero probability while state $\bar{v}$ has non-zero probability. When it is player 1's turn, state X has non-zero probability and the player-2-action states have zero probability. When it is player 2's turn, exactly one of the player-2-action states has non-zero probability. When the game is over, state "done" has non-zero probability and all others zero, and when the game starts, state "start" has non-zero probability and all others zero.

Initially, the probability of each of the states is 0, except for the "start" state, which has probability 1. The "start" action causes a stochastic transition that results in the literal-related states representing the initial assignment α, as well as a transition to either X or all the player-2-action states. The start action results in negative reward from all other states, meaning that it will only be chosen as the first action. Figure F.1 depicts the POMDP state space, and the transitions, rewards, and observations for the "start" action.

Player 1 can take three types of moves, the simplest of which is to toggle a single X variable. The $tog(x)$ action causes the probability in states x and $\bar{x}$ to swap. In addition, a stochastic transition is made from state X to the player-2-action states; each of these states has a unique observation associated with it, and therefore only one will have non-zero probability after an observation is made. From the literal-related states, any of the player-2-action observations can be made; as a result, they have zero probability after a transition if and only if they had zero probability before the transition. To prevent player 1 from choosing out of turn, negative rewards are issued for a $tog(x)$ action from all of the player-2-action states. Figure F.2 depicts the rewards, transitions, and observations associated with the $tog(x)$ actions. The second type of move, *pass*, is implemented similarly, except that it may also be issued when it is player 2's turn.

In the third type of move, player 1 can simultaneously toggle a variable *and* declare a win. Recall that player 1 wins if the 13-DNF formula is true after that player's move. For a 13-DNF formula to be true, at least one of the terms of the formula must be true and all 13 of the literals in that term must be true. We define action $togwin(x,t)$ to toggle the value of variable $x \in X$ and declare that term t of the formula is true. Given any $togwin(x,t)$ action, all states make a transition to "done" (the game is over). Once again, precautions are taken to ensure that player 1 moves in turn, however, additional constraints force player 1 to take a $togwin(x,t)$ action only when it results in a win. This is done by placing a negative reward on every state associated with a literal that appears *negated* in term t (except for x which has a negative reward directly, because its toggled value is considered in making the decision as to whether or not the term is satisfied), and thus the action has zero reward if and only if all the literals in term t are true.

Since player 2 is not explicitly represented in the POMDP, we must force the agent (player 1) to make player 2's moves on its behalf. To do this, we introduce $tog(y)$ and $challenge(y,t,l)$ actions for each $y \in Y$, $t \in F$, and literal l. Once again, we use negative rewards to ensure that the agent takes a player-2 action when it is player 2's turn. Player 2's choice of action is represented by the single player-2-action state that has non-zero probability. For action $tog(y)$, there are negative rewards out of state X and the other player-2-action states; this makes $tog(y)$ the only action choice possible when state $tog(y)$ has non-zero probability.

The $challenge(y,t,l)$ actions are a bit different. When state $togwin(y,t)$ has non-zero probability, this means that player 2 claims that toggling the value of variable y results in a win for player 2 because term t becomes true. Player 1's response is to choose a literal in term t that proves that term t is *not* true. For each y, t, l triple, $challenge(y,t,l)$ results in a transition to "done" with a negative reward on the

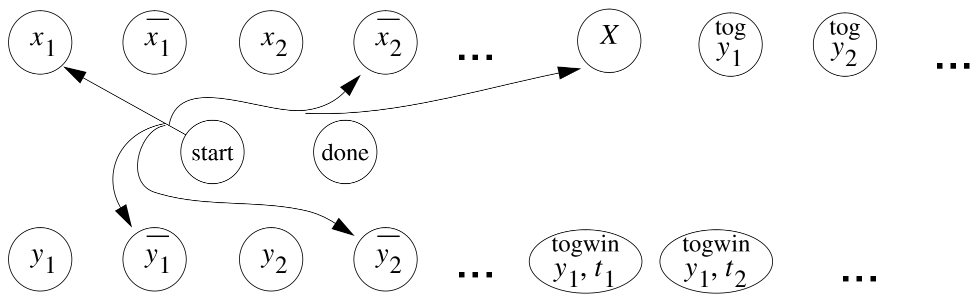

Figure F.1: Transitions for the "start" action. All states with no outgoing transition have negative-reward transitions. All observations among the player-2-action states are distinct. The union of these observations is possible from each of the other states.

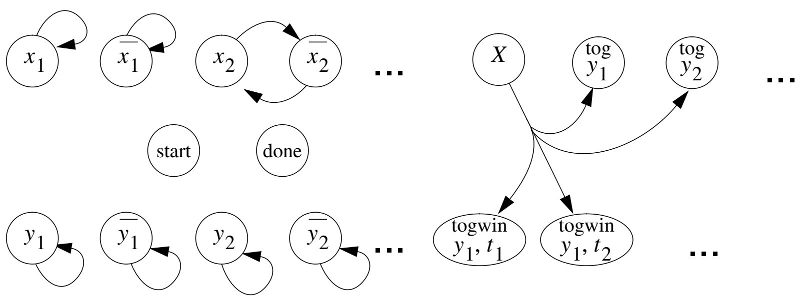

Figure F.2: Transitions for the $tog(x_2)$ action. All states with no outgoing transition have negative-reward transitions. All observations among the player-2-action states are distinct. The union of these observations is possible from each of the other states.

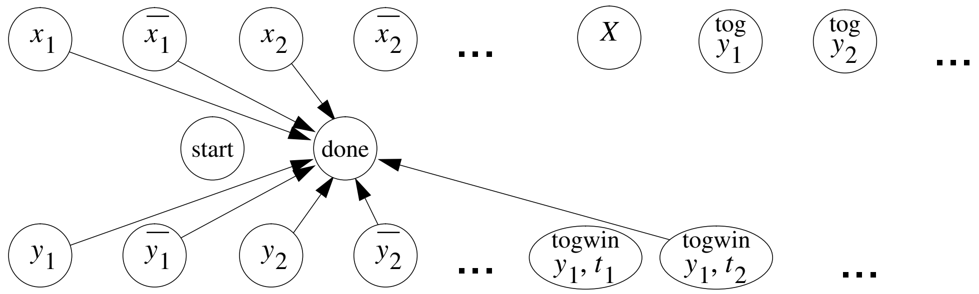

Figure F.3: Transitions for the *challenge*$(y_1, t_2, \bar{x_2})$ action. All states with no outgoing transition have negative-reward transitions. All observations are equal.

transition from literal l; this means that if l is actually true, player 1 loses. It is in player 1's best interest to choose a literal that proves that term t is not satisfied (if such a literal exists). Figure F.3 depicts the rewards, transitions, and observations associated with the *challenge*(y, t, l) actions.

Given this construction, there is a tight analogy between reachable configurations of states in the POMDP and legal states of the game. In particular, if player 2 has a win, there is no policy that will prevent the agent from reaching a negative reward. Conversely, if player 1 has a win or a draw, then player 1 has a choice of actions that result in only zero rewards forever. Thus, a procedure for deciding whether a POMDP has a zero-reward optimal policy can be used to decide the winner in the boolean-formula game, after an easy transformation.

Because the boolean-formula game is known to EXPTIME-hard, solving boolean-reward POMDPs is EXPTIME-hard. It is known that P≠EXPTIME and that there are problems in EXPTIME that truly take exponentially long to solve. Unlike the NP and PSPACE-hard problems described earlier, which are likely to be intractable, POMDPs are *provably* intractable.

F.3 A Difficult POMDP For Q-learning

This section briefly describes an example POMDP, constructed by Chrisman, for which a naive Q-learning algorithm learns the worst possible policy.

The POMDP is illustrated in Figure F.4 and consists of 3 states, 3 actions, and a single observation. The discount factor is 0.9 and state s_1 is the start state. An optimal policy in this environment is to take action a_1, and then to alternate between actions

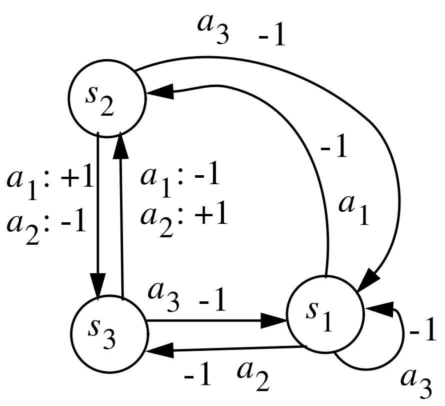

Figure F.4: A hard POMDP for Q-learning.

a_1 and a_2. This policy has an immediate reward of -1 on the first step, then $+1$ on every step thereafter.

There are three deterministic memoryless policies for this environment: always a_1, always a_2, and always a_3. In the long run, always a_1 and always a_2 have the same reward, alternating between $+1$ and -1 immediate reward.

Will now consider the behavior of a simple Q-learning agent in this environment. The agent's internal data structure consists of a Q function mapping sensations to actions, therefore it is a vector of three values: $Q[a_1], Q[a_2]$, and $Q[a_3]$. Let us initialize the Q values to be all zeros, and use a learning rate of 0.1. Action selection is greedy (choose the action with the highest Q value), with ties broken according to the order: a_1, a_2, a_3.

Here is what happens when the Q-learner faces Chrisman's POMDP: It starts with Q values all the same. By the tie-breaking scheme, action a_1 is chosen in state s_1 to start. The immediate reward of -1 makes state s_1 look the worst. Now the agent is in state s_2 with a tie between actions a_2 and a_3. The agent chooses action a_2. This results in an immediate reward of -1 for action a_2, and makes action a_3 the best remaining choice. After taking action a_3 in state s_3, the agent returns to state s_1 with all Q values tied once again. Thus, actions a_1, a_2, and a_3 are deterministically selected in that order resulting in immediate rewards of -1 at every step. No policy achieves worse performance in this environment.

This example is fairly robust to changes in the learning rate and the discount factor. It is not robust to changes in the action selection scheme, either by choosing actions non-greedily or breaking ties differently. Nevertheless, it illustrates that a simple Q-learning algorithm can perform miserably on a simple POMDP.

Appendix G

Supplementary Results on Information-state MDPs

In this appendix, I prove several fundamental results concerning information-state Markov decision processes. I show that it is not difficult to compute the Bellman error magnitude when value functions are represented by policy trees, that useful policy trees can be identified easily, that there are complicated one-stage POMDPs, that solving one-stage POMDPs is NP-complete under randomized reductions, and that witnesses can be identified easily.

G.1 Computing the Bellman Error Magnitude

One method for stopping value iteration in an information-state MDP is to wait for the maximum difference between consecutive value functions V_t and V_{t-1} to be less than some ϵ. The maximum difference between value functions, called the Bellman error magnitude, is easily computed for finite-state-space models. In this section, I examine two algorithms for computing the Bellman error magnitude in information-state MDPs when the value functions are represented as sets of policy trees Γ_t and Γ_{t-1}.

The first algorithm is exact and somewhat expensive, while the second is a bound but very cheap to compute. Both run in polynomial time. The section is not intended as an exhaustive account of possible algorithms; instead, it simply serves to show that efficient algorithms for this problem exist.

G.1.1 An Exact Method

Briefly, the exact method considers all pairs of policy trees $p_t \in \Gamma_t$ and $p_{t-1} \in \Gamma_{t-1}$. It uses linear programming to find an information state x^* such that

1. p_t dominates the policy trees in Γ_t at x^*,
2. p_{t-1} dominates the policy trees in Γ_{t-1} at x^*, and
3. $\delta = |V_{p_t}(x^*) - V_{p_{t-1}}(x^*)|$ is maximized over all x^* satisfying the first two conditions.

The first two conditions guarantee that $V_{p_t}(x^*) = V_t(x^*)$ and $V_{p_{t-1}}(x^*) = V_{t-1}(x^*)$. The third condition makes δ a lower bound on the largest difference between value functions. By finding the pair of policy trees that give the largest value for δ, the Bellman error magnitude is identified.

An algorithm that uses this idea to compute the Bellman error magnitude appears in Table G.1. To compute the maximum absolute value, two separate linear programs are constructed.

G.1.2 A Bound

The previous section described a method for computing the maximum difference between two piecewise-linear and convex value functions represented as sets of policy trees. This section describes a simpler approach that is much more efficient to compute but only gives an upper bound on the difference.

We want to bound the biggest difference between two value functions, $V_t(x) = \max_{p \in \Gamma_t} V_p(x)$ and $V_{t-1}(x) = \max_{p \in \Gamma_{t-1}} V_p(x)$. The following lemma gives an inexpensive way to bound the biggest positive difference between V_t and V_{t-1}.

Lemma G.1 *Let*

$$\delta = \max_{p_t \in \Gamma_t} \min_{p_{t-1} \in \Gamma_{t-1}} \max_{s \in \mathcal{S}} (V_{p_t}(s) - V_{p_{t-1}}(s)).$$

Then, for all information states x, $V_t(x) - V_{t-1}(x) \leq \delta$.

Proof: Consider some particular information state x. Let $p_t^* = \mathrm{argmax}_{p \in \Gamma_t} V_p(x)$, $p_{t-1}^* = \mathrm{argmax}_{p \in \Gamma_{t-1}} V_p(x)$, and

$$p'_{t-1} = \operatorname*{argmin}_{\tilde{p} \in \Gamma_{t-1}} \max_{s \in \mathcal{S}} (V_{p_t^*}(s) - V_{\tilde{p}}(s)).$$

```
BellmanErrMag(Γ_t, Γ_{t-1}) := {
    maxdiff := -∞
    foreach p_t ∈ Γ_t, p_{t-1} ∈ Γ_{t-1} {
        δ_1 := checkpair(p_{t-1}, Γ_{t-1}, p_t, Γ_t)
        δ_2 := checkpair(p_t, Γ_t, p_{t-1}, Γ_{t-1})
        maxdiff := max{maxdiff, δ_1, δ_2}
    }
    return maxdiff
}

checkpair(p, Γ, p', Γ') := {
    Solve the following linear program:
        maximize: δ
        s.t.: Σ_s x[s]V_p(s) ≥ Σ_s x[s]V_p̃(s), for all p̃ ∈ Γ - {p}
        and: Σ_s x[s]V_p'(s) ≥ Σ_s x[s]V_p̃(s), for all p̃ ∈ Γ' - {p'}
        and: δ = Σ_s x[s]V_p(s) - Σ_s x[s]V_p'(s),
        and: Σ_s x[s] = 1
        and: x[s] ≥ 0, for all s ∈ S
        variables: δ, x[s] for all s ∈ S
    if (δ = undefined) then return -∞
    else return δ
}
```

Table G.1: Subroutine for computing the exact Bellman error magnitude in polynomial time.

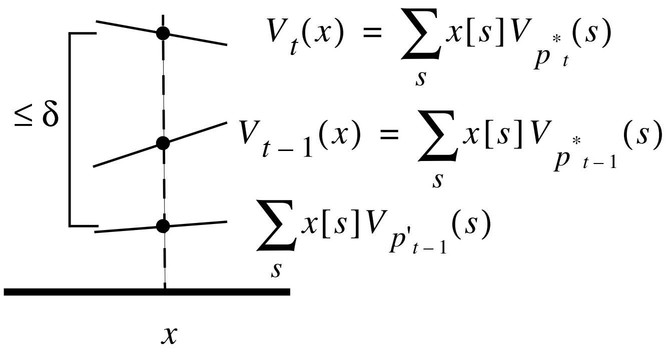

Figure G.1: Upper bound on the maximum difference between value functions.

In words, p_t^* is the best policy tree in Γ_t at x, p_{t-1}^* is the best policy tree in Γ_{t-1} at x, and p'_{t-1} is the policy tree whose value function has the smallest possible biggest difference from that of p_t^* at any state. Define δ as in the statement of the lemma. This situation is depicted in Figure G.1.

We can bound

$$\begin{aligned}
V_t(x) - V_{t-1}(x) &= V_{p_t^*}(x) - V_{p_{t-1}^*}(x) \\
&\le V_{p_t^*}(x) - V_{p'_{t-1}}(x) = \sum_{s\in\mathcal{S}} x[s]V_{p_t^*}(s) - \sum_{s\in\mathcal{S}} x[s]V_{p'_{t-1}}(s) \\
&\le \sum_{s\in\mathcal{S}} x[s](V_{p_t^*}(s) - V_{p'_{t-1}}(s)) \\
&\le \max_{s\in\mathcal{S}}(V_{p_t^*}(s) - V_{p'_{t-1}}(s)) = \min_{p_{t-1}\in\Gamma_{t-1}} \max_{s\in\mathcal{S}}(V_{p_t^*}(s) - V_{p_{t-1}}(s)) \\
&\le \max_{p_t\in\Gamma_t} \min_{p_{t-1}\in\Gamma_{t-1}} \max_{s\in\mathcal{S}}(V_{p_t}(s) - V_{p_{t-1}}(s)) = \delta,
\end{aligned}$$

as desired. Since x was chosen arbitrarily, the bound holds for all $x \in \mathcal{X}$. □

Lemma G.1 gives a one-way bound on the Bellman error magnitude. A more complete bound can be found by reversing the roles of Γ_t and Γ_{t-1} in the lemma and combining the result with the bound from Lemma G.1.

Although the bound obtained this way can be arbitrarily loose, it is a good approximation in the following sense. If the policy trees in Γ_t are identical to the policy trees in Γ_{t-1}, the bound will (correctly) state that the Bellman error magnitude is zero. And, although it is difficult to formalize, if the two sets are only slightly different, the given bound will be fairly accurate.

Table G.2 provides a subroutine for the upper bound.

```
BellmanErrMagBound(Γ_t, Γ_{t-1}) := {
    δ_1 := max_{p_t∈Γ_t} min_{p_{t-1}∈Γ_{t-1}} max_{s∈S} (V_{p_t}(s) − V_{p_{t-1}}(s))
    δ_2 := max_{p_{t-1}∈Γ_{t-1}} min_{p_t∈Γ_t} max_{s∈S} (V_{p_{t-1}}(s) − V_{p_t}(s))
    return max{δ_1, δ_2}
}
```

Table G.2: Computing a bound on the Bellman error magnitude.

G.2 Identifying Useful Policy Trees

In this section, I prove Lemma 7.1, which states that the useful policy trees are precisely those that dominate the other policy trees for some information state. To show this, I begin by proving Lemma 7.2.

Given an information state x and a set of policy trees F, let p^* be the lexicographic maximum policy tree in F such that $V_{p^*}(x) = V_t(x)$. Lemma 7.2 asserts the existence of an x' such that p^* dominates all other policy trees in F at x'.

To show this formally, we need some additional notation. Information state x is a *ρ-step from x_1 towards x_2* if $x = \rho(x_2 - x_1) + x_1$, $0 \leq \rho \leq 1$. Let e_s be the vector corresponding to the "corner" of $\mathcal{X}$ where all the probability mass is on state $s \in \mathcal{S}$.

Lemma G.2 *Let $W = \{p \in F | V_p(x) \geq V_{p'}(x) \text{ for all } p' \in F\}$; that is, W is the set of policy trees in F that dominate those in $F - W$ at x. For all $s \in \mathcal{S}$, if $x \neq e_s$, there is a value $\rho > 0$ such that for the information state x' that is a ρ-step from x towards e_s, the policy trees in W still dominate those in $F - W$.*

Proof: For any $p \in W$ and any $p' \in F - W$, define $\Delta = V_p(x) - V_{p'}(x) > 0$. Let $x' = \rho(e_s - x) + x$. We can find a value for $\rho > 0$ such that V_p is bigger than $V_{p'}$ at x'. Using the linearity of the value function for policy trees, the following statements are equivalent:

$$\begin{aligned}
V_p(x') &> V_{p'}(x') \\
(V_p(\rho(e_s - x) + x) - V_{p'}(\rho(e_s - x) + x) &> 0 \\
(1-\rho)(V_p(x) - V_{p'}(x)) + \rho(V_p(e_s) - V_{p'}(e_s) &> 0 \\
(1-\rho)\Delta + \rho(V_p(s) - V_{p'}(s)) &> 0 \\
\rho(\Delta + V_{p'}(s) - V_p(s)) &< \Delta
\end{aligned}$$

Since $\Delta > 0$, we can divide by it without changing the inequality. Let $\kappa = 1 + (V_{p'}(s) - V_p(s))/\Delta$. Then the above expression is equivalent to $\rho\kappa < 1$. If $\kappa \leq 0$, the inequality holds with $\rho \leq 1$. Otherwise, we can set $\rho \leq 1/(2\kappa)$ to satisfy the inequality.

This shows that for every s, we can find a $\rho > 0$ for each pairing of $p \in W$ and $p' \in F - W$ such that a ρ-step from x to e_s gives an x' such that $V_p(x') > V_{p'}(x')$. Since there are a finite number of ways of pairing elements in W with those in $F - W$, there is a $\rho > 0$ that works for *all* pairs (namely, the minimum ρ for any pair). $\square$

We can now use Lemma G.2 to prove Lemma 7.2. The plan is to take successive ρ-steps from x towards each corner e_s. Each step is small enough so that the lexicographic maximum policy tree in W still dominates the policy trees in $F - W$, but large enough so that some ties within W are broken.

Let $W_0 = W$ be the set of policy trees maximal at $x_0 = x$. By Lemma G.2, there is an information state, x_1, strictly different from x_0 if $x_0 \neq e_{s_0}$ and in the direction of e_{s_0} at which the policy trees in W_0 are still bigger than the others. Let W_1 be a subset of W_0 consisting of the policy trees maximal at x_1. If $x_0 = e_{s_0}$, let $W_1 = W_0$.

It should be clear that the policy trees in W_1 are precisely those $p \in W_0$ for which $V_p(s_0) = V_{p^*}(s_0)$, that is, those tied with the lexicographically maximal policy tree in the first component.

If we apply this argument inductively for each component, W_i becomes the set of all policy trees in W that agree with V_{p^*} in the first i components of their value functions. The policy trees in W_i dominate those in $F - W_i$ at x_i. After every state has been considered, we are left with $W_{|\mathcal{S}|} = \{p^*\}$ with p^* as the unique policy tree dominating all others in F at $x_{|\mathcal{S}|}$.

This concludes the proof of Lemma 7.2. In addition to its computational importance, Lemma 7.2 is also helpful for proving Lemma 7.1, which states that a policy tree is useful if and only if there is an information state for which it dominates all other policy trees.

Let G be a set of policy trees and $V(x) = \max_{p \in G} V_p(x)$ be the value function they represent. Let Γ be the set of policy trees in G that dominate the other policy trees in G for some information state. I first show that every policy tree $p \in \Gamma$ is necessary for representing V. Consider any x for which p dominates the policy trees in G. Such an x exists by construction of Γ. From the definition of domination, we know that no other policy tree in G gives as large a value at x as p does. Therefore, the value function represented by any set $U \subseteq G - \{p\}$ would have a strictly smaller value at x than V does. Thus, every $p \in \Gamma$ is needed to represent V.

Next, we show that Γ is a sufficient representation of V, that is,

$$V(x) = \max_{p \in \Gamma} V_p(x).$$

To see this, consider any $x \in \mathcal{X}$. Let p^* be the lexicographic maximum policy tree in G such that $V_{p^*}(x) = \max_{p \in G} V_p(x)$. By Lemma 7.2, there exists an x' such that p^* dominates all other policy trees in G at x', so p^* will be included in Γ. Thus, $V(x) = \max_{p \in G} V_p(x) = V_{p^*}(x) = \max_{p \in \Gamma} V_p(x)$. Since this holds for every x, Γ represents the same function that G does.

G.3 Example One-stage POMDP Problems

In this section, I provide several constructions that illustrate various important POMDP issues. I will start with a lemma that makes it easier to describe specific one-stage POMDP problems.

Recall that to specify a one-stage POMDP problem, it is necessary to define sets $\mathcal{S}$, $\mathcal{Z}$, and $\mathcal{A}$; functions T, O, and R; a scalar β; and a set Γ of $|\mathcal{S}|$-vectors. The set Γ' is the minimum set of policy trees such that

$$\max_{p \in \Gamma'} V_p(x) = \max_{a \in A} \max_{\tau \in \mathcal{T}(\mathcal{Z} \to \Gamma)} \sum_s x[s] \left(R(s,a) + \beta \sum_z \sum_{s'} T(s,a,s') O(s',a,z) \tau(z)[s'] \right), \tag{G.1}$$

for all $x \in \mathcal{X}$, where $\mathcal{T}(\mathcal{Z} \to \Gamma)$ is the set of all functions mapping $\mathcal{Z}$ to Γ. The set Γ' of $|\mathcal{S}|$-vectors can be viewed as a representation of a piecewise-linear convex function $f(x) = \max_{p \in \Gamma'} V_p(x)$ over the $(|\mathcal{S}| - 1)$-dimensional simplex. Because Γ' is defined via a one-stage POMDP problem, the $\mathcal{Z}$, $\mathcal{A}$, T, O, R, β, and Γ quantities can be viewed as a specification of a piecewise-linear convex function.

I will show that there are values for $\mathcal{Z}$, $\mathcal{A}$, T, O, R, β, and Γ such that the piecewise-linear convex function they specify via Γ' has particular properties. I show that by specifying a finite set K of $|\mathcal{S}|$-vectors, and a finite collection Λ of pairs of $|\mathcal{S}|$-vectors with components between zero and one, it is possible to specify a piecewise-linear convex function that is the solution to a one-stage POMDP problem, without needing to identify $\mathcal{Z}$, $\mathcal{A}$, T, O, R, β, and Γ directly.

Let K be a set of $|\mathcal{S}|$-vectors. Let $\mathcal{V}$ be a finite set of *variables*, and $\mathcal{B}$ be the set of all *bindings* mapping the elements of $\mathcal{V}$ to $\{1,2\}$ ($\mathcal{B} = \mathcal{T}(\mathcal{V} \to \{1,2\})$). Let Λ be a set of $|\mathcal{S}|$-vectors, $\Lambda = \{\lambda_k^z | z \in \mathcal{V}, k \in \{1,2\}\}$. Let the sets K and Λ specify a piecewise-linear convex function f over $\mathcal{X}$ by

$$f(x) = \max\left\{\max_{\sigma \in K}(x \cdot \sigma), \max_{b \in \mathcal{B}}(x \cdot \alpha_b)\right\},$$

where $\alpha_b = \sum_{z \in \mathcal{V}} \lambda_{b(z)}^z$ for $b \in \mathcal{B}$.

The next lemma shows that any piecewise-linear convex function that can be specified this way can also be specified as the solution to a one-stage POMDP problem of similar size.

Lemma G.3 *Given a set of variables $\mathcal{V}$, and sets K and Λ of $|\mathcal{S}|$-vectors ($|\mathcal{S}| \geq |\mathcal{V}|+2$), and a set $\Lambda = \{\lambda_k^z | z \in \mathcal{V}, k \in \{1,2\}\}$ ($0 \leq \lambda_k^z[s] \leq 1, \forall s \in \mathcal{S}$), there is a one-stage* POMDP *problem such that*

$$\max_{p \in \Gamma'} V_p(x) = \max\left\{\max_{\sigma \in K}(x \cdot \sigma), \max_{b}(x \cdot \alpha_b)\right\}.$$

The POMDP *has $|\mathcal{Z}| = |\mathcal{V}|$, $|\mathcal{A}| = |K| + 1$, and $|\Gamma| = 2$.*

Proof: Define the one-stage POMDP as follows. The set of observations is the set of variables ($\mathcal{Z} = \mathcal{V}$), and there is one action for each vector in K and one corresponding to the Λ set, $\mathcal{A} = \{a_\sigma | \sigma \in K\} \cup \{a_0\}$. Let Γ be a set of two $|\mathcal{S}|$-vectors, γ_1 and γ_2, and let $\beta = 1$. Define γ_1 and γ_2 by

$$\gamma_k[s'] = \begin{cases} (3-2k)(|\mathcal{Z}|+1), & \text{if } s' \leq |\mathcal{Z}|, \\ (k-1)|\mathcal{Z}|(|\mathcal{Z}|+1), & \text{if } s' = |\mathcal{Z}|+1, \\ 0, & \text{otherwise,} \end{cases}$$

for $k \in \{1,2\}$.

Let each a_σ action result in an immediate reward of $x \cdot \sigma$ and a transition to state $|\mathcal{Z}|+2$ (which has zero value under Γ); for all s, $R(a_\sigma, s) = \sigma[s]$, $T(s, a_\sigma, s') = 1$, $O(s', a_\sigma, z) = 1/|\mathcal{Z}|$ if $s' = |\mathcal{Z}|+2$ and zero otherwise. Since $\gamma_k[|\mathcal{Z}|+2] = 0$ for $k \in \{1,2\}$, for all $\tau \in \mathcal{T}(\mathcal{Z} \to \{\gamma_1, \gamma_2\})$ and $s \in \mathcal{S}$

$$R(s, a_\sigma) + \beta \sum_z \sum_{s'} T(s, a_\sigma, s') O(s', a_\sigma, z) \tau(z)[s'] = R(s, a_\sigma) = \sigma[s]. \tag{G.2}$$

The a_0 action is a bit more complex. Define the POMDP functions as follows,

$$
\begin{aligned}
R(s,a_0) &= \sum_z \frac{\lambda_1^z[s]+\lambda_2^z[s]-1}{2}, \\
T(s,a_0,s') &= \begin{cases} \frac{\lambda_1^{s'}[s]-\lambda_2^{s'}[s]+1}{2(|\mathcal{Z}|+1)}, & \text{if } s' \le |\mathcal{Z}|, \\ \frac{1}{|\mathcal{Z}|+1}, & \text{if } s' = |\mathcal{Z}|+1, \\ 1-\sum_{s'=1}^{|\mathcal{Z}|+1} T(s,a_0,s'), & \text{if } s' = |\mathcal{Z}|+2, \\ 0, & \text{otherwise.} \end{cases}, \\
O(s',a_0,z) &= \begin{cases} 1, & \text{if } z = s', \\ 0, & \text{if } s' \le |\mathcal{Z}| \text{ and } z \ne s', \\ \frac{1}{|\mathcal{Z}|}, & \text{otherwise.} \end{cases},
\end{aligned}
$$

It is not hard to show that the components of T and O add to 1 in the proper way.

For any $s \in \mathcal{S}$, $\tau \in \mathcal{T}(\mathcal{Z} \to \Gamma)$, let b be the binding such that $\tau(z) = \gamma_{b(z)}$ for all $z \in \mathcal{Z}$, then,

$$
\begin{aligned}
&R(s,a_0) + \beta \sum_z \sum_{s'} T(s,a_0,s') O(s',a_0,z) \tau(z)[s'] \\
&= R(s,a_0) + \beta \sum_z \sum_{s'} T(s,a_0,s') O(s',a_0,z) \gamma_{b(z)}[s'] \\
&= \sum_z \frac{\lambda_1^z[s]+\lambda_2^z[s]-1}{2} + \sum_z \frac{\lambda_1^z[s]-\lambda_2^z[s]+1}{2(|\mathcal{Z}|+1)} \gamma_{b(z)}[z] + \sum_z \frac{1}{|\mathcal{Z}|+1} \frac{1}{|\mathcal{Z}|} \gamma_{b(z)}[|\mathcal{Z}|+1] \\
&= \sum_z \Bigg(\frac{\lambda_1^z[s]+\lambda_2^z[s]-1}{2} + \frac{\lambda_1^z[s]-\lambda_2^z[s]+1}{2(|\mathcal{Z}|+1)} (3-2b(z))(|\mathcal{Z}|+1) \\
&\qquad + \frac{1}{|\mathcal{Z}|+1} \frac{1}{|\mathcal{Z}|} (b(z)-1)|\mathcal{Z}|(|\mathcal{Z}|+1) \Bigg) \\
&= \sum_z \left(\frac{\lambda_1^z[s]+\lambda_2^z[s]-1}{2} + \frac{\lambda_1^z[s]-\lambda_2^z[s]+1}{2} (3-2b(z)) + (b(z)-1) \right) \\
&= \sum_z \left(\frac{\lambda_1^z[s](4-2b(z)) + \lambda_2^z[s](-2+2b(z)) + 2 - 2b(z)}{2} + (b(z)-1) \right) \\
&= \sum_z (\lambda_1^z[s](2-b(z)) + \lambda_2^z[s](-1+b(z)) + 1 - b(z) + b(z) - 1) \\
&= \sum_z \lambda_{b(z)}^z[s] = \alpha_b[s]. \qquad \text{(G.3)}
\end{aligned}
$$

Combining Equations G.1, G.2, and G.3, we have

$$
\begin{aligned}
&\max_{p\in\Gamma'} V_p(x)\\
&= \max_{a\in A}\max_{\tau\in\mathcal{T}(\mathcal{Z}\to\Gamma)}\sum_s x[s]\left(R(s,a)+\beta\sum_z\sum_{s'}T(s,a,s')O(s',a,z)\tau(z)[s']\right)\\
&= \max\left\{\max_{\tau\in\mathcal{T}(\mathcal{Z}\to\Gamma)}\sum_s x[s]\left(R(s,a_0)+\beta\sum_z\sum_{s'}T(s,a_0,s')O(s',a_0,z)\tau(z)[s']\right),\right.\\
&\qquad\left.\max_{\sigma\in K}\max_{\tau\in\mathcal{T}(\mathcal{Z}\to\Gamma)}\sum_s x[s]\left(R(s,a_\sigma)+\beta\sum_z\sum_{s'}T(s,a_\sigma,s')O(s',a_\sigma,z)\tau(z)[s']\right)\right\}\\
&= \max\left\{\max_{b\in\mathcal{B}}(x\cdot\alpha_b),\max_{\sigma\in K}(x\cdot\sigma)\right\};
\end{aligned}
$$

therefore, the supplied vectors correspond exactly to the set of value functions obtained by solving a one-stage POMDP problem.

□

G.3.1 Exponential Number of Useful Policy Trees

I show that there exists a family of one-stage POMDPs such that, for every $n > 2$, $|\mathcal{S}| = 2n$, $|\mathcal{A}| = 1$, $|\mathcal{Z}| = n$, $|\Gamma_{t-1}| = 2$, and $|\Gamma_t| = 2^n$.

Let $\mathcal{S} = \{1,\dots,2n\}$, and $V = \{1,\dots,n\}$. For $k \in \{1,2\}$, let $\lambda_k^z[2z+k-2] = 1$, and 0 otherwise. Let $K = \emptyset$. By Lemma G.3, there is a POMDP with $|\mathcal{A}| = 1$, $|\mathcal{Z}| = n$, and $|\Gamma| = 2$ such that $G = \bigcup_{b\in B}\{\sum_z \lambda^z_{b(z)}\}$. Note that $|G| = |B| = 2^{|V|} = 2^n$. We know that the set Γ' is a subset of G. We can show that, in fact, these sets are equal, because every policy tree in G dominates all other policy trees in G at some $x \in \mathcal{X}$.

Lemma G.4 *In the construction just described, for every $b \in B$ there is an $x^* \in \mathcal{X}$ such that $x^*\cdot\alpha_b > x^*\cdot\alpha_{b'}$ for all $b' \neq b$.*

Proof: For a binding b, for each $z \in V$, let $x^*[2z+b(z)-2] = 1/n$ and 0 otherwise.

The components of x^* are non-negative and sum to 1, so $x^* \in \mathcal{X}$. Now, for any $b' \in B$,

$$\begin{aligned} x^* \cdot \alpha_{b'} &= x^* \cdot \sum_z \lambda^z_{b'(z)} \\ &= \sum_s (x^*[s] \sum_z \lambda^z_{b'(z)}[s]) \\ &= \sum_z \sum_s (x^*[s] \lambda^z_{b'(z)}[s]) \\ &= \sum_z x^*[2z + b'(z) - 2] \\ &= \frac{1}{n} I\{b'(z) = b(z)\}, \end{aligned}$$

which is uniquely maximized for $b' = b$. □

G.3.2 Exponential Number of Vertices in a Region

In this section, I show that there is a family of one-stage POMDP problems such that, for every $n > 2$, $|\mathcal{S}| = n+1$, $|\mathcal{A}| = 2n+1$, $|\mathcal{Z}| = 1$, $|\Gamma| = 1$, $|\Gamma'| \leq 2n+1$, and the number of vertices in the region dominated by one of the policy trees is 2^n.

Once again, I will use the specification described in Lemma G.3, inductively creating a separate piecewise-linear convex function for each n. For each n, define a set K_n containing $2n+1$ vectors, and let σ^n be a vector such that the vectors in K_n form the walls of an n-dimensional hypercube bounding the region $\{x | x \cdot \sigma^n \geq x \cdot \sigma, \sigma \in K_n\}$. We define the family of POMDPs recursively, starting with $n = 1$. Let $\mathcal{S}_1 = \{1, 2\}$ and $\mathcal{V} = \{1\}$. Let $\lambda^z_k[s] = 0$ for all $z \in \mathcal{V}$, $k \in \{1, 2\}$, and $s \in \mathcal{S}$ (the set Λ is not needed in this construction). The notation $\langle x, y \rangle$ concatenates two vectors (or a vector and scalar) into a single vector.

Let $K_1 = \{\langle 1, -1/2 \rangle, \langle 0, 0 \rangle, \langle -1, 0 \rangle\}$ and $\sigma^1 = \langle 0, 0 \rangle$. Let Ω_n be the set of vertices of the region $\{x | x \cdot \sigma^n \geq x \cdot \sigma, \sigma \in K_n\}$. The vertices are the information states where one of the other vectors in K_1 gives the same value as σ^1. It is easy to verify that the set of vertices $\Omega_1 = \{\langle 0, 1 \rangle, \langle 1/3, 2/3 \rangle\}$.

Inductively define $\sigma^{n+1} = \langle \sigma^n, 0 \rangle$,

$$K_{n+1} = \{\langle \sigma, 0 \rangle : \sigma \in K_n\} \cup \{\langle 0, \ldots, 0, -1 \rangle, \langle 1, \ldots, 1, -1 \rangle\}$$

and $\mathcal{S}_{n+1} = \mathcal{S}_n \cup \{n+2\}$.

Note that $\sigma^{n+1} = \langle 0, \ldots, 0 \rangle \in K_{n+1}$. The vertices of σ^{n+1}'s region are the information states where $n+1$ of the vectors in K_{n+1} give the same value as σ^{n+1}. Let

$x \in \Omega_n$. Inductively, n vectors in K_n give the same value as σ^n at x. Let σ be any one of these vectors. It follows from the construction of σ that

$$\langle x, 0\rangle \cdot \langle \sigma, 0\rangle = x \cdot \sigma = 0 = x \cdot \sigma^n = \langle x, 0\rangle \cdot \sigma^{n+1}$$

and

$$\langle x, 0\rangle \cdot \langle 0, \ldots, 0, -1\rangle = 0 = \langle x, 0\rangle \cdot \sigma^{n+1},$$

so $\langle x, 0\rangle \in X$ is a vertex of σ^{n+1}'s region. Similarly,

$$\langle 1/2\, x, 1/2\rangle \cdot \langle \sigma, 0\rangle = 1/2\, x \cdot \sigma + 0 = 0 = \langle 1/2\, x, 1/2\rangle \cdot \sigma^{n+1}$$

and

$$\langle 1/2\, x, 1/2\rangle \cdot \langle 1, \ldots, 1, -1\rangle = 1/2 - 1/2 = 0 = \langle 1/2\, x, 1/2\rangle \cdot \sigma^{n+1},$$

so $\langle 1/2\, x, 1/2\rangle \in X$ is a vertex of σ^{n+1}'s region also. As a result,

$$\Omega_{n+1} = \{\langle x, 0\rangle : x \in \Omega_n\} \cup \{\langle 1/2\, x, 1/2\rangle : x \in \Omega_n\}.$$

Since all the vectors in K_{n+1} and Ω_{n+1} are unique, $|K_{n+1}| = |K_n| + 2$, and $|\Omega_{n+1}| = 2|\Omega_n|$, $|K_n| = 2n+1$ and $|\Omega_n| = 2^n$ for all $n \geq 2$. Applying Lemma G.3, there is a one-stage POMDP problem for each $n > 2$ with $|\mathcal{S}| = n+1$, $|\mathcal{A}| = 2n+1$, $|\mathcal{Z}| = 1$, $|\Gamma| = 1$, $|\Gamma'| = n+1$, such that the number of vertices in one of the linear regions of the value function is 2^n.

G.4 Solving One-stage POMDP Problems is Hard

In this section, I show that the problem of solving polynomially output-bounded one-stage POMDP problems is NP-complete under randomized reductions. This means that there is a randomized algorithm for solving one-stage POMDP problems in polynomial time if and only if RP=NP. To show this, we examine a deep connection between this problem and the *unique-satisfying-assignment problem*, defined below.

A boolean formula in conjunctive normal form (CNF) is an "and" of a set of clauses of "ors" of literals (variables and negated variables). A *satisfying assignment* maps each of the variables to either "true" or "false" so the entire formula evaluates to "true." There is a result, proved by Valiant and Vazirani [165], that implies that there exists a polynomial-time algorithm for finding a satisfying assignment for a formula that is guaranteed to have at most one satisfying assignment only if RP=NP[1]. I will show that a polynomial-time algorithm for solving polynomially output-bounded POMDPs could be used to solve the unique-satisfying-assignment problem in polynomial time, and therefore that such an algorithm exists only if RP=NP.

I use the specification described in Lemma G.3 and define $\mathcal{S}$, $\mathcal{V}$, K, and Λ. Take a CNF formula consisting of a set of $M > 1$ clauses C, and variables $\mathcal{V}$ ($|\mathcal{V}| \geq 2$). The set of variables corresponds both to the variables in Lemma G.3 and the boolean variables in the formula. Let $\mathcal{S} = C \times \mathcal{V}$. An element of $\mathcal{S}$ is a pair $(c, z) \in C \times \mathcal{V}$. There is a pair of λ vectors for each variable $z \in \mathcal{V}$, which will encode the CNF formula. Vector λ_1^z indicates in which clauses variable z is unnegated and λ_2^z indicates in which clauses variable z is negated. More specifically, for each $z \in \mathcal{V}$ and $k \in \{1, 2\}$, λ_k^z is a $|C \times \mathcal{V}|$-vector with $\lambda_1^z[(c, z)] = 1$ if variable z appears unnegated in clause c and $\lambda_2^z[(c, z)] = 1$ if variable z appears negated in clause c. All other components of the λ vectors are zero.

For a binding $b \in \mathcal{B}$, $b(z) = 1$ if variable z is true in the assignment and $b(z) = 2$ otherwise. Thus, if $\lambda_{b(z)}^z[(c, z)] = 1$, then clause c evaluates to "true" under binding b because of the binding of variable z in that clause. Let κ be a $|C \times \mathcal{V}|$-vector with each component equal to $(M - 1/2)/M$. For each $c \in C$, define a $|C \times \mathcal{V}|$-vector σ_c, as follows. For all $z \in \mathcal{V}$, let $\sigma_c[(c, z)] = 1 + (M - 1/2)/M - M$ and $\sigma_c[(c', z)] = 1 + (M - 1/2)/M$ for $c' \neq c$. As described earlier, let $\alpha_b = \sum_z \lambda_{b(z)}^z$. From the definition of λ, every component of α_b is either a one or a zero.

Each α_b vector corresponds to a variable assignment and the σ_c and κ vectors are designed to jointly dominate all possible α_b vectors, except those corresponding to satisfying assignments.

Lemma G.5 *Let $K = \{\kappa\} \cup \{\sigma_c | c \in C\}$. There is an $x^* \in \mathcal{X}$ such that $x^* \cdot \alpha_b > \max_{\sigma \in K}(x^* \cdot \sigma)$ if and only if b is a satisfying assignment.*

Proof: First, assume b is a satisfying assignment. We can construct an x^* such that $x^* \cdot \alpha_b = 1$, but $x^* \cdot \kappa < 1$ and $x^* \cdot \sigma_c < 1$ for all $c \in C$ as follows.

For each clause $c \in C$, pick a single variable z_c such that the binding of that variable in b causes clause c to be satisfied. Let $x^*[(c, z_c)] = 1/M$ for each $c \in C$ and 0 otherwise. Note that $x^* \in \mathcal{X}$ because M components are set to $1/M$. Because of the zeros in x^*

[1] Thanks to Avrim Blum for pointing this out.

and the λ vectors,

$$\begin{aligned}
x^* \cdot \alpha_b &= \sum_c \sum_z x^*[(c,z)] \sum_{z'} \lambda^{z'}_{b(z')}[(c,z)] \\
&= \sum_c \sum_z x^*[(c,z)] \lambda^{z}_{b(z)}[(c,z)] \\
&= \sum_c x^*[(c,z_c)] \lambda^{z_c}_{b(z_c)}[(c,z_c)] = \sum_c \frac{1}{M} = 1.
\end{aligned}$$

Using the same x^* we see that $x^* \cdot \kappa = (M - 1/2)/M < 1$, and for each clause $c \in C$,

$$\begin{aligned}
x^* \cdot \sigma_c &= \sum_z \sum_{c'} x^*[(c',z)] \sigma_c[(c',z)] \\
&= \sum_z \left(x^*[(c,z)] \left(1 + \frac{M-\frac{1}{2}}{M} - M\right) + \sum_{c' \neq c} x^*[(c',z)] \left(1 + \frac{M - \frac{1}{2}}{M}\right)\right) \\
&= \sum_z \left(-M x^*[(c,z)] + \sum_{c'} x^*[(c',z)] \left(1 + \frac{M - \frac{1}{2}}{M}\right)\right) \\
&= -M\frac{1}{M} + \left(1 + \frac{M - \frac{1}{2}}{M}\right) \sum_z \sum_{c'} x^*[(c',z)] \\
&= -1 + 1 + \frac{M - \frac{1}{2}}{M} = \frac{M - \frac{1}{2}}{M} < 1.
\end{aligned}$$

Thus, $x^* \cdot \alpha_b = 1 > \max_{\sigma \in K}(x^* \cdot \sigma)$, for satisfying assignment b.

Now I show that, for a non-satisfying b, $x \cdot \alpha_b < \max_{\sigma \in K}(x \cdot \sigma)$ for all $x \in \mathcal{X}$. We proceed by contradiction. Assume we have an $x \in \mathcal{X}$ such that $\max_{\sigma \in K}(x \cdot \sigma) \leq x \cdot \alpha_b$ for a non-satisfying b. Then, $x \cdot \kappa \leq x \cdot \alpha_b$ and $x \cdot \sigma_c \leq x \cdot \alpha_b$ for all $c \in C$. Define $w_c = \sum_z x[(c,z)]$. The variable w_c is the *weight* of clause c, and note that $\sum_c w_c = 1$ if $x \in \mathcal{X}$. Let $c^* = \operatorname{argmin}_c w_c$; c^* is a minimum weight clause.

If $x \cdot \kappa \leq x \cdot \alpha_b$, then $(M - 1/2)/M \leq x \cdot \alpha_b = x \cdot [\sum_z \lambda^z_{b(z)}] \leq 1 - w_{c^*}$. The last inequality is justified by the fact that b is not satisfying so at least one clause contributes zero to the dot product and assuming it is the smallest weight clause gives us the largest possible value. This restricts w_{c^*} so that

$$w_{c^*} \leq \frac{1}{2M} < \frac{1}{2M} + \frac{1}{2(M-1)} = \frac{M - \frac{1}{2}}{M(M-1)}. \tag{G.4}$$

At the same time, it must be the case that $x \cdot \sigma_{c^*} \leq x \cdot \alpha_b$, which implies $1 + (M - 1/2)/M - Mw_{c^*} \leq 1 - w_{c^*}$ and therefore $w_{c^*} \geq (M - 1/2)/(M(M-1))$. This directly contradicts Inequality G.4, and therefore we can conclude that, for a non-satisfying b, $x \cdot \alpha_b < \max_{\sigma \in K}(x \cdot \sigma)$ for all $x \in \mathcal{X}$. $\square$

By Lemma G.3, there is a POMDP such that

$$\max_{\gamma\in\Gamma'}(x\cdot\gamma) = \max\left\{\max_{\sigma\in K}(x\cdot\sigma), \max_{b}(x\cdot\alpha_b)\right\}.$$

This one-stage POMDP problem is derived from the unique-satisfying-assignment-problem instance and has the property that $\Gamma' \not\subseteq K$ if and only if the boolean-formula instance is satisfiable. Because of the assumption that the boolean formula has no more than 1 satisfying assignment, $|\Gamma'| \leq M+2$; thus, the one-stage POMDP problem is polynomially output bounded.

Because the one-stage POMDP problem instance can be created in polynomial time, and the condition $\Gamma' \not\subseteq K$ can be checked in polynomial time, a polynomial-time algorithm for solving polynomially output-bounded one-stage POMDP problems could be used to find unique satisfying assignments in polynomial time. As mentioned at the start of this section, this would imply RP=NP.

To complete the proof of Theorem 7.1, I need to argue that if RP=NP then there is a randomized polynomial-time algorithm for solving polynomially output-bounded one-stage POMDP problems. We can build up a set of vectors $U \subseteq \Gamma'$ one at a time by answering the question "Is there an information state x such that $\max_{\gamma\in U} x\cdot\gamma \neq V_t(x)$?" and adding the dominating vector at x into U. For polynomially output-bounded one-stage POMDP problems, if the answer to this question is yes, then there is an x that can be written using polynomially many bits. Such an x can be identified in nondeterministic polynomial time using standard techniques, and therefore in randomized polynomial time if RP=NP. Because there are at most a polynomial number of vectors that can be added to U, the process terminates with $U = \Gamma'$ in polynomial time.

G.5 Proof of the Witness Lemma

Let U be a set of useful policy trees for action a. In this section, I show that the set U does not equal the complete set Γ^a of useful policy trees if and only if some policy tree, in the set of neighbors of policy trees in U, dominates the policy trees in U.

The "if" direction is easy since the neighbor can be used to identify a policy tree missing from U.

The "only if" direction can be rephrased as: If $U \neq \Gamma^a$ then there is an information state $x \in \mathcal{X}$, a policy tree $p \in U$, and a neighbor p' of p such that p' dominates all

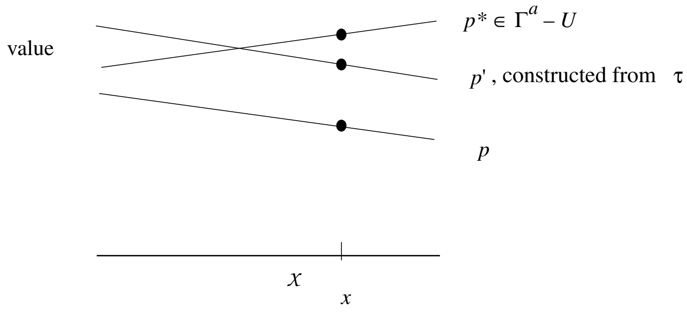

Figure G.2: An illustration of some of the quantities used in Theorem 7.3.

policy trees in U at x. Figure G.2 illustrates some of the relevant quantities used to show this.

Start by picking $p^* \in \Gamma^a - U$ and choose any x such that p^* dominates the policy trees in U at x. Let $p = \text{argmax}_{\tilde{p}\in U} V_{\tilde{p}}(x)$. As illustrated in the figure, p^* is the optimal policy tree at x, and p is the best policy tree in U at x. By construction, $V_{p^*}(x) > V_p(x)$.

If p^* and p are neighbors, we are done, since we are searching for a neighbor of p that dominates the other policy trees in U at x, and p^* meets these requirements.

If p^* and p are not neighbors, we will identify a neighbor p' of p that does satisfy these requirements. Choose an observation $z^* \in \mathcal{Z}$ such that

$$x \cdot \mathbf{stval}(a, z^*, p^*) > x \cdot \mathbf{stval}(a, z^*, p).$$

There must be a z^* satisfying this inequality since otherwise we get the contradiction

$$\begin{aligned} V_{p^*}(x) &= \sum_s x[s] \left(R(s,a) + \beta \sum_z \mathbf{stval}(a, z, p^*)[s] \right) \\ &\leq \sum_s x[s] \left(R(s,a) + \beta \sum_z \mathbf{stval}(a, z, p)[s] \right) = V_p(x). \end{aligned}$$

Let $\tau \in \mathcal{T}(\mathcal{Z} \to \Gamma)$ where $\tau(z^*) = \mathbf{subtree}(p^*, z^*)$ and $\tau(z) = \mathbf{subtree}(p, z)$ for $z \neq z^*$. Let p' be the policy tree constructed from τ, $p' = \mathbf{tree}(a, \tau)$. By construction, p and

p' are neighbors. In addition,

$$\begin{aligned} V_{p'}(x) &= \sum_s x[s] \left(R(s,a) + \beta \sum_z \mathbf{stval}(a,z,p')[s] \right) \\ &= \sum_s x[s] \left(R(s,a) + \beta \sum_{z \neq z^*} \mathbf{stval}(a,z,p)[s] + \beta\ \mathbf{stval}(a,z,p^*)[s] \right) \\ &> \sum_s x[s] \left(R(s,a) + \beta \sum_z \mathbf{stval}(a,z,p)[s] \right) = V_p(x) = \max_{\tilde{p} \in U} V_{\tilde{p}}(x). \end{aligned}$$

Therefore the policy tree p' dominates all policy trees in U at x.

Bibliography

[1] N. Abe and M. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.

[2] David H. Ackley and Michael L. Littman. Generalization and scaling in reinforcement learning. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 550–557, San Mateo, California, 1990. Morgan Kaufmann.

[3] David H. Ackley and Michael L. Littman. Interactions between learning and evolution. In C. Langton, C. Taylor, J. D. Farmer, and S. Ramussen, editors, *Artificial Life II: Santa Fe Institute Studies in the Sciences of Complexity*, volume 10, pages 487–509. Addison-Wesley, Redwood City, California, 1991.

[4] Aristotle Arapostathis, Vivek S. Borkar, Emmanuel Fernández-Gaucherand, Mrinal K. Ghosh, and Steven I. Marcus. Discrete-time controlled Markov processes with average cost criterion: A survey. *SIAM Journal on Control and Optimization*, 31(2):282–344, March 1993.

[5] K. J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.

[6] Yuri Bahturin. *Basic Structures of Modern Algebra.* Kluwer Academic Publishers, Norwell, Massachusetts, 1993.

[7] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37, San Francisco, California, 1995. Morgan Kaufmann.

[8] Leemon C. Baird and A. H. Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical Report WL-TR-93-1147, Wright-Patterson Air Force Base Ohio: Wright Laboratory, 1993.

[9] Andrew G. Barto, S. J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[10] Andrew G. Barto, Richard S. Sutton, and Christopher J. C. H. Watkins. Learning and sequential decision making. Technical Report 89-95, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, 1989. Also published in *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, Michael Gabriel and John Moore, editors. The MIT Press, Cambridge, Massachusetts, 1991.

[11] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathmatical Statistics*, 41(1):164–171, 1970.

[12] R. Beckers, O. E. Holland, and J. L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In Rodney A. Brooks and Pattie Maes, editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 181–189, Cambridge, Massachusetts, 1994. Bradford Books/MIT Press.

[13] Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

[14] Y. Bengio and P. Frasconi. An input/output HMM architecture. In G. Tesauro, D. S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 427–434, Cambridge, Massachusetts, 1995. The MIT Press.

[15] Dimitri P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[16] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, Massachusetts, 1995. Volumes 1 and 2.

[17] Dimitri P. Bertsekas and David A. Castañon. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34(6):589–598, June 1989.

[18] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[19] David Blackwell. Discrete dynamic programming. *Annals of Mathematical Statistics*, 33(2):719–726, June 1962.

[20] R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107, May 1977.

[21] Jim Blythe. Planning with external events. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 1994.

[22] Craig Boutilier. Imposed and learned conventions in multiagent decision processes: Extended abstract. Unpublished manuscript, 1995.

[23] Craig Boutilier, Thomas Dean, and Steve Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Second European Workshop on Planning*, 1995.

[24] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[25] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. Unpublished manuscript, 1995.

[26] Justin A. Boyan. Modular neural networks for learning context-dependent game strategies. Master's thesis, Department of Engineering and Computer Laboratory, University of Cambridge, Cambridge, United Kingdom, August 1992.

[27] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 671–678. Morgan Kaufmann, San Mateo, California, 1994.

[28] Justin A. Boyan and Andrew W. Moore. Algorithms for approximating optimal value functions in acyclic domains. Unpublished manuscript, 1995.

[29] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, Cambridge, Massachusetts, 1995. The MIT Press.

[30] Justin A. Boyan, Andrew W. Moore, and Richard S. Sutton. Proceedings of the workshop on value function approximation, Machine Learning Conference 1995. Technical Report CMU-CS-95-206, Carnegie Mellon University, School of Computer Science, 1995.

[31] Dima Burago, Michel de Rougemont, and Anatol Slissenko. On the complexity of partially observed Markov decision processes. *Theoretical Computer Science*, to appear.

[32] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, 1994.

[33] Hsien-Te Cheng. *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, British Columbia, Canada, 1988.

[34] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188, San Jose, California, 1992. AAAI Press.

[35] Dave Cliff and Susi Ross. Adding temporary memory to ZCS. *Adaptive Behavior*, 3(2):101–150, 1994.

[36] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, February 1992.

[37] Anne Condon. On algorithms for simple stochastic games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 13:51–71, 1993.

[38] Jonathan Connell and Sridhar Mahadevan. Rapid task learning for real robots. In *Robot Learning*, pages 105–140. Kluwer Academic Publishers, Boston, Massachusetts, 1993.

[39] Don Coppersmith and Schmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of 19th Annual ACM Symposium on Theory of Computing*, pages 1–6, 1987.

[40] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[41] George Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.

[42] Peter Dayan and Terrence J. Sejnowski. Exploration bonuses and dual control. *Machine Learning*, to appear.

[43] Thomas Dean, Leslie Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1-2):35–74, 1995.

[44] Eric V. Denardo. *Dynamic Programming: Models and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[45] F. D'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10:98–108, 1963.

[46] Cyrus Derman. *Finite State Markovian Decision Processes*. Academic Press, New York, New York, 1970.

[47] David P. Dobkin and Steven P. Reiss. The complexity of linear programming. *Theoretical Computer Science*, 11:1–18, 1980.

[48] A. W. Drake. *Observation of a Markov Process Through a Noisy Channel*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1962.

[49] Denise Draper, Steve Hanks, and Dan Weld. Probabilistic planning with information gathering and contingent execution. Technical Report 93-12-04, University of Washington, Seattle, Washington, December 1993.

[50] Denise Draper, Steve Hanks, and Daniel Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, pages 76–82, 1994.

[51] James N. Eagle. The optimal search for a moving target when the search path is constrained. *Operations Research*, 32(5):1107–1115, 1984.

[52] E. Fernández-Gaucherand, M. K. Ghosh, and S. I. Marcus. Controlled Markov processes on the infinite planning horizon: Weighted and overtaking cost criteria. Technical Report TR 93-6, The University of Maryland, 1995.

[53] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971. Reprinted in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate, eds., Morgan Kaufmann, 1990.

[54] J. A. Filar. Ordered field property for stochastic games when the player who controls transitions changes from state to state. *Journal of Optimization Theory and Applications*, 34:503–515, 1981.

[55] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, California, 1979.

[56] J. C. Gittins. *Multi-armed Bandit Allocation Indices*. Wiley-Interscience series in systems and optimization. Wiley, Chichester, New York, 1989.

[57] Leslie M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, pages 25–29, 1977.

[58] Geoffrey J. Gordon. Stable function approximation in dynamic programming. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 261–268, San Francisco, California, 1995. Morgan Kaufmann.

[59] Vijaykumar Gullapalli and Andrew G. Barto. Convergence of indirect adaptive asynchronous value iteration algorithms. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 695–702, San Mateo, California, 1994. Morgan Kaufmann.

[60] Eric A. Hansen. Completely observable Markov decision processes with observation costs. Unpublished manuscript, 1995.

[61] Mance E. Harmon, Leemon C. Baird, III, and Harry Klopf. Reinforcement learning applied to a differential game. *Adaptive Behavior*, 4(1):3–28, 1995.

[62] Matthias Heger. Consideration of risk in reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 105–111, San Francisco, California, 1994. Morgan Kaufmann.

[63] Matthias Heger. The loss from imperfect value functions in expectation-based and minimax-based tasks. *Machine Learning*, to appear.

[64] O. Hernández-Lerma and S. I. Marcus. Adaptive control of discounted Markov decision chains. *Journal of optimization theory and applications*, 46(2):227–235, June 1985.

[65] O. Hernandez-Lerma and S. I. Marcus. Adaptive control of Markov processes with incomplete state information and unknown parameters. *Journal of Optimization Theory and Applications*, 52(2):227–241, February 1987.

[66] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Compuation.* Addison-Wesley, Redwood, California, 1991.

[67] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12:359–370, 1966.

[68] Ronald A. Howard. *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge, Massachusetts, 1960.

[69] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6), November 1994.

[70] Tommi Jaakkola, Satinder Pal Singh, and Michael I. Jordan. Monte-carlo reinforcement learning in non-Markovian decision problems. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, Cambridge, Massachusetts, 1995. The MIT Press.

[71] George H. John. When the best move isn't optimal: Q-learning with exploration. Unpublished manuscript, 1995.

[72] Leslie Pack Kaelbling. *Learning in Embedded Systems.* The MIT Press, Cambridge, Massachusetts, 1993.

[73] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. Technical Report CS-96-08, Brown University, Providence, Rhode Island, 1995.

[74] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, to appear.

[75] G. Kalai. A subexponential randomized simplex algorithm. In *Proceedings of 24th Annual ACM Symposium on the Theory of Computing*, pages 475–482, 1992.

[76] L. C. M. Kallenberg. *Linear Programming and Finite Markovian Control Problems.* Number 148 in Mathematical Centre Tracts. Mathematisch Centrum, Amsterdam, 1983.

[77] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the American Society of Mechanical Engineers, Journal of Basic Engineering*, 82:35–45, March 1960.

[78] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[79] L. G. Khachian. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

[80] Victor Klee. On the number of vertices of a convex polytope. *Canada Journal of Mathematics*, XVI:701–720, 1964.

[81] Victor Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities, III*, pages 159–175. Academic Press, New York, 1972.

[82] Daphne Koller and Nimrod Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4:528–552, 1992.

[83] Daphne Koller and Nimrod Megiddo. Finding mixed strategies with small supports in extensive form games. *International Journal of Game Theory*, to appear.

[84] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on the Theory of Computing*, pages 750–759, 1994.

[85] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, to appear.

[86] P. R. Kumar and P. P. Varaiya. *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[87] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, September 1995.

[88] Harold J. Kushner and A. J. Kleinman. Mathematical programming and the control of Markov chains. *International Journal of Control*, 13(5):801–820, 1971.

[89] Long-Ji Lin and Tom M. Mitchell. Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, School of Computer Science, May 1992.

[90] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, San Francisco, California, 1994. Morgan Kaufmann.

[91] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Cambridge, Massachusetts, 1994. The MIT Press.

[92] Michael L. Littman. The witness algorithm: Solving partially observable Markov decision processes. Technical Report CS-94-40, Brown University, Department of Computer Science, Providence, Rhode Island, December 1994.

[93] Michael L. Littman and David H. Ackley. Adaptation in constant utility nonstationary environments. In Rik K. Belew and Lashon Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 136–142, San Mateo, California, 1991. Morgan Kaufmann.

[94] Michael L. Littman, Anthony Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, San Francisco, California, 1995. Morgan Kaufmann.

[95] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Efficient dynamic-programming updates in partially observable Markov decision processes. Technical Report CS-95-19, Brown University, Providence, Rhode Island, 1996.

[96] Michael L. Littman, Thomas L. 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)*, Montreal, Québec, Canada, 1995.

[97] Michael L. Littman and Csaba Szepesvári. A generalized reinforcement-learning model: Convergence and applications. Technical Report CS-96-10, Brown University, Providence, Rhode Island, 1996.

[98] Michael Lederman Littman. Algorithms for sequential decision making. Technical Report CS-96-09, Brown University, March 1996.

[99] William S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39(1):162–175, January–February 1991.

[100] William S. Lovejoy. A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, 28:47–66, 1991.

[101] Walter Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117:151–155, 1995.

[102] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, to appear.

[103] Andrew Kachites McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, December 1995.

[104] R. Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 190–196, Amherst, Massachusetts, 1993. Morgan Kaufmann.

[105] R. Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 387–395, San Francisco, California, 1995. Morgan Kaufmann.

[106] Lisa Meeden, G. McGraw, and D. Blank. Emergent control and planning in an autonomous vehicle. In D.S. Touretsky, editor, *Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society*, pages 735–740. Lawerence Erlbaum Associates, Hillsdale, New Jersey, 1993.

[107] Mary Melekopoglou and Anne Condon. On the complexity of the policy iteration algorithm for stochastic games. Technical Report CS-TR-90-941, Computer Sciences Department, University of Wisconsin Madison, 1990. To appear in the ORSA Journal on Computing.

[108] Nicolas Meuleau. *Exploration or Exploitation? Real-time learning.* PhD thesis, Universite de Caen, forthcoming.

[109] George E. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28:1–16, January 1982.

[110] P.R. Montague and T.J. Sejnowski. The predictive brain: Temporal coincidence and temporal order in synaptic learning mechanisms. *Learning and Memory*, 1(1):1–33, 1994.

[111] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13, 1993.

[112] Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. *Machine Learning*, 21, 1995.

[113] Ann Nicholson and Leslie Pack Kaelbling. Toward approximate planning in very large stochastic domains. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, Stanford, California, 1994.

[114] Guillermo Owen. *Game Theory: Second edition.* Academic Press, Orlando, Florida, 1982.

[115] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[116] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, August 1987.

[117] Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[118] Azaria Paz. *Introduction to Probabilistic Automata.* Academic Press, New York, 1971.

[119] Jing Peng and Ronald J. Williams. Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 1(4):437–454, 1993.

[120] H. J. M. Peters and O. J. Vrieze, editors. *Surveys in game theory and related topics.* Number 39 in CWI Tract. Stichting Mathematisch Centrum, Amsterdam, 1987.

[121] Loren K. Platzman. *Finite-memory Estimation and Control of Finite Probabilistic Systems.* PhD thesis, Massachusetts Institute of Technology, 1977.

[122] Loren K. Platzman. Optimal infinite-horizon undiscounted control of finite probabilistc systems. *SIAM Journal of Control and Optimization*, 18:362–380, 1980.

[123] Loren K. Platzman. A feasible computational approach to infinite-horizon partially-observed Markov decision problems. Technical report, Georgia Institute of Technology, Atlanta, Georgia, January 1981.

[124] M. L. Puterman and S. L. Brumelle. The analytic theory of policy iteration. In Martin L. Puterman, editor, *Dynamic Programming and its applications*, pages 91–114. Academic Press, New York, New York, 1978.

[125] M. L. Puterman and S. L. Brumelle. On the convergence of policy iteration in stationary dynamic programming. *Mathematics of Operations Research*, 4:60–69, 1979.

[126] Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, New York, 1994.

[127] Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted Markov decision processes. *Management Science*, 24:1127–1137, 1978.

[128] Eric Rasmusen. *Games and Information: An Introduction to Game Theory*. Oxford, New York, New York, 1989.

[129] Mark B. Ring. *Continual Learning in Reinforcement Environments*. PhD thesis, University of Texas at Austin, Austin, Texas, August 1994.

[130] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[131] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error backpropagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition. Volume 1: Foundations*, chapter 8. The MIT Press, Cambridge, Massachusetts, 1986.

[132] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, New Jersey, 1994.

[133] John Rust. Numerical dynamic programming in economics. In *Handbook of Computational Economics*. Elsevier, North Holland, 1996.

[134] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:211–229, 1959. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York 1963.

[135] Katsushige Sawaki and Akira Ichikawa. Optimal control for partially observable Markov decision processes over an infinite horizon. *Journal of the Operations Research Society of Japan*, 21(1):1–14, March 1978.

[136] Robert E. Schapire and Manfred K. Warmuth. On the worst-case analysis of temporal-difference learning algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 266–274, San Francisco, California, 1994. Morgan Kaufmann.

[137] Jürgen H. Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 500–506, San Mateo, California, 1991. Morgan Kaufmann.

[138] Marcel J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence 10*, pages 1039–1046, 1987.

[139] Nicol N. Schraudolph, Peter Dayan, and Terrence J. Sejnowski. Temporal difference learning of position evaluation in the game of Go. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 817–824, San Mateo, California, 1994. Morgan Kaufmann.

[140] Alexander Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience, New York, New York, 1986.

[141] Anton Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 298–305, Amherst, Massachusetts, 1993. Morgan Kaufmann.

[142] Tom Seaver. *How I Would Pitch to Babe Ruth.* Playboy Press, Chicago, Illinois, 1974.

[143] L.S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39:1095–1100, 1953.

[144] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995.

[145] Satinder Pal Singh. *Learning to Solve Markovian Decision Processes.* PhD thesis, Department of Computer Science, University of Massachusetts, 1993. Also, CMPSCI Technical Report 93-77.

[146] Satinder Pal Singh, Tommi Jaakkola, and Michael I. Jordan. Model-free reinforcement learning for non-Markovian decision problems. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 284–292, San Francisco, California, 1994. Morgan Kaufmann.

[147] Satinder Pal Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16, 1994.

[148] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[149] Edward Sondik. *The Optimal Control of Partially Observable Markov Processes.* PhD thesis, Stanford University, 1971.

[150] Edward J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2), 1978.

[151] Larry J. Stockmeyer and Ashok K. Chandra. Provably difficult combinatorial games. *SIAM Journal of Computing*, 8(2):151–174, May 1979.

[152] Gilbert Strang. *Linear Algebra and its Applications: Second Edition.* Academic Press, Orlando, Florida, 1980.

[153] Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1984.

[154] Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[155] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, Texas, 1990. Morgan Kaufmann.

[156] Richard S. Sutton. Planning by incremental dynamic programming. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 353–357. Morgan Kaufmann, 1991.

[157] Csaba Szepesvári. General framework for reinforcement learning. In *Proceedings of ICANN'95 Paris*, 1995.

[158] Csaba Szepesvári and Michael L. Littman. Generalized Markov decision processes: Dynamic-programming and reinforcement-learning algorithms. Technical Report CS-96-11, Brown University, Providence, Rhode Island, 1996.

[159] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, pages 58–67, March 1995.

[160] Gerald J. Tesauro. Practical issues in temporal difference. In J. E. Moody, S. J. Hanson, and D. S. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 259–266, San Mateo, California, 1992. Morgan Kaufmann.

[161] Sebastian Thrun. Learning to play the game of chess. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, Cambridge, Massachusetts, 1995. The MIT Press.

[162] Paul Tseng. Solving H-horizon, stationary Markov decision problems in time proportional to $\log(H)$. *Operations Research Letters*, 9(5):287–297, 1990.

[163] John N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3), September 1994.

[164] John N. Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, to appear.

[165] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(1):85–93, 1986.

[166] J. van der Wal. *Stochastic Dynamic Programming.* Number 139 in Mathematical Centre tracts. Mathematisch Centrum, Amsterdam, 1981.

[167] Sergio Vendu and H. Vincent Poor. Abstract dynamic programming models under commutativity conditions. *SIAM Journal of Control and Optimization*, 25(4):990–1006, July 1987.

[168] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, New Jersey, 1947.

[169] Koos Vrieze. Zero-sum stochastic games. In H. J. M. Peters and O. J. Vrieze, editors, *Surveys in game theory and related topics*, pages 103–132. Stichting Mathematisch Centrum, Amsterdam, 1987.

[170] O. J. Vrieze. *Stochastic games with finite state and action spaces*. Number 33 in CWI Tract. Stichting Mathematisch Centrum, Amsterdam, 1987.

[171] O. J. Vrieze and S. H. Tijs. Fictitious play applied to sequences of games and discounted stochastic games. *International Journal of Game Theory*, 11(2):71–85, 1982.

[172] K.-H. Waldmann. On bounds for dynamic programs. *Mathematics of Operations Research*, 10(2):220–232, May 1985.

[173] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, United Kingdom, 1989.

[174] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[175] C. C. White and D. Harrington. Application of Jensen's inequality for adaptive suboptimal design. *Journal of Optimization Theory and Applications*, 32(1):89–99, 1980.

[176] Chelsea C. White, III. Partially observed Markov decision processes: A survey. *Annals of Operations Research*, 32, 1991.

[177] Chelsea C. White, III and William T. Scherer. Solution procedures for partially observed Markov decision processes. *Operations Research*, 37(5):791–797, September-October 1989.

[178] Chelsea C. White, III and William T. Scherer. Finite-memory suboptimal design for partially observed Markov decision processes. *Operations Research*, 42(3):439–455, May-June 1994.

[179] Steven D. Whitehead and Long-Ji Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1-2):271–306, February 1995.

[180] Ronald J. Williams and Leemon C. Baird, III. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, Northeastern University, College of Computer Science, Boston, Massachusetts, November 1993.

[181] Stewart Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):147–173, 1995.

[182] Stewart W. Wilson. Knowledge growth in an artificial animal. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 16–23, Hillsdale, New Jersey, 1985. Lawrence Erlbaum Associates.

[183] Uri Zwick and Mike S. Paterson. The complexity of mean payoff games. *Theoretical Computer Science*, to appear.