Incremental Dynamic Programming for On-Line Adaptive Optimal Control

Steven J. Bradtke

CMPSCI Technical Report 94-62

August 1994

INCREMENTAL DYNAMIC PROGRAMMING FOR ON–LINE ADAPTIVE OPTIMAL CONTROL

A Dissertation Presented

by

STEVEN J. BRADTKE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1994

Computer Science

INCREMENTAL DYNAMIC PROGRAMMING FOR ON–LINE ADAPTIVE OPTIMAL CONTROL

A Dissertation Presented

by

STEVEN J. BRADTKE

To
my wife,
Gloria,
and to our children,
Thomas and Molly

ACKNOWLEDGEMENTS

I wish to thank Dr. Andrew G. Barto for the many contributions he made to this dissertation. The work described here grew out of many discussions on the theory and practice of Reinforcement Learning, Dynamic Programming, and function approximation. The intellectual climate he fostered was instrumental to my growth as a scientist, and his personal support and encouragement helped me through several trying periods.

I would like to give special thanks to Dr. B. Erik Ydstie, my guru on the theory of Linear Quadratic Regulation. My discussion of the application of Reinforcement Learning to adaptive optimal Linear Quadratic control would have been considerably weaker without his extensive and always cheerful help.

The Adaptive Networks Laboratory was always a pleasant and intellectually stimulating workplace, due to the high caliber of my fellow staff members. I would like to especially thank the following, who helped make my years in the group so rewarding: Richard Yee, Mike Duff, Bob Crites, Jonathan Bachrach, Satinder Singh, Vijaykumar Gullapalli, Neil Berthier, Robbie Jacobs, and Brian Pinette.

Finally, I would like to thank Glo, Tommy, and Molly for their patience and understanding as I trudged slowly to the completion of this dissertation. They make everything possible.

ABSTRACT

INCREMENTAL DYNAMIC PROGRAMMING FOR ON–LINE ADAPTIVE OPTIMAL CONTROL

SEPTEMBER 1994

STEVEN J. BRADTKE

B.S., MICHIGAN STATE UNIVERSITY
M.S., UNIVERSITY OF MICHIGAN
Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew G. Barto

Reinforcement learning algorithms based on the principles of Dynamic Programming (DP) have enjoyed a great deal of recent attention both empirically and theoretically. These algorithms have been referred to generically as Incremental Dynamic Programming (IDP) algorithms. IDP algorithms are intended for use in situations where the information or computational resources needed by traditional dynamic programming algorithms are not available. IDP algorithms attempt to find a global solution to a DP problem by incrementally improving local constraint satisfaction properties as experience is gained through interaction with the environment. This class of algorithms is not new, going back at least as far as Samuel's adaptive checkers-playing programs, but the links to DP have only been noted and understood very recently.

This dissertation expands the theoretical and empirical understanding of IDP algorithms and increases their domain of practical application. We address a number of issues concerning the use of IDP algorithms for on-line adaptive optimal control. We present a new algorithm, Real-Time Dynamic Programming, that generalizes Korf's Learning Real-Time A* to a stochastic domain, and show that it has computational advantages over conventional DP approaches to such problems. We then describe several new IDP algorithms based on the theory of Least Squares function approximation. Finally, we begin the extension of IDP theory to continuous domains by considering the problem of Linear Quadratic Regulation. We present an algorithm based on Policy Iteration and Watkins' Q-functions and prove convergence of the algorithm (under the appropriate conditions) to the optimal policy. This is the first result proving convergence of a DP-based reinforcement learning algorithm to the optimal policy for any continuous domain. We also demonstrate that IDP algorithms cannot be applied blindly to problems from continous domains, even such simple domains as Linear Quadratic Regulation.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v
ABSTRACT vi
LIST OF TABLES x
LIST OF FIGURES xi

Chapter

1. INTRODUCTION 1

1.1 Several Challenges for Incremental Dynamic Programming 1

1.1.1 On-line convergence 1
1.1.2 Continuous Problems 2
1.1.3 Issues of Representation and Function Approximation 3
1.1.4 Model-based vs. Model-free methods 3
1.1.5 Hidden state information 5

1.2 Organization of the Dissertation 5

2. MARKOV DECISION PROBLEMS AND DYNAMIC PROGRAMMING 7

2.1 Markov Decision Problems 7

2.1.1 The grid world 9
2.1.2 Backgammon 11

2.2 Dynamic Programming 11

2.2.1 Greedy Policies 13
2.2.2 Synchronous DP 13

2.2.2.1 Evaluating a policy 14
2.2.2.2 Value Iteration 15
2.2.2.3 Policy Iteration 15

2.2.3 Asynchronous DP 16
2.2.4 Temporal Difference Methods 18

3. REAL-TIME DYNAMIC PROGRAMMING 20

3.1 State–Space Search 20
3.2 Learning Real-Time A* 22
3.3 Stochastic Shortest Path Problems 23
3.4 Real-time Dynamic Programming 24
3.5 Adaptive Real-time Dynamic Programming 26
3.6 Real-Time Q-learning 28
3.7 Exploration 28
3.8 Experiments 30

3.8.1 The Race Track Problem 30
3.8.2 Results 32

3.9 Iterative Deepening Real-time Dynamic Programming 41
3.10 Conclusions 45

4. NEW ALGORITHMS FOR TEMPORAL DIFFERENCE LEARNING 47

4.1 The TD(λ) learning rule 47
4.2 Previous convergence results for TD(λ) 48
4.3 Probability 1 convergence of on-line TD(0) 50
4.4 The Normalized TD(λ) learning rule 51
4.5 A Least Squares approach to TD learning 52

4.5.1 Linear Least Squares function approximation 52
4.5.2 Algorithm LS TD 54
4.5.3 Convergence of algorithm LS TD 56
4.5.4 Proof of Theorem 4.4 58
4.5.5 Proof of Theorem 4.5 59
4.5.6 Algorithm RLS TD 59
4.5.7 Dependent or extraneous features 60

4.6 A related algorithm of Werbos 61
4.7 Choosing an algorithm 63
4.8 The TD error variance 64
4.9 Experiments 64

4.9.1 Selecting learning rates 67

4.10 Conclusions 67

5. TEMPORAL DIFFERENCE METHODS FOR LINEAR QUADRATIC SYSTEMS 69

5.1 Linear Quadratic Regulation 70
5.2 Q-functions and Policy improvement 71
5.3 Direct Estimation of Q-functions 72
5.4 Adaptive Policy Iteration for LQR 73
5.5 Demonstration of the adaptive policy iteration algorithm in simulation 78

5.6 Comparison with model-based adaptive optimal control 81
5.7 The Failure of Optimizing Q-learning for LQR 83
5.8 Adaptive Optimal Control of the Input-Output Behavior 85
5.9 Distributed Adaptive Control 90
5.10 Dealing with a Stochastic System 94
5.11 Conclusions 95

6. CONCLUSIONS 97

APPENDICES

A. PROOF OF THEOREMS FROM CHAPTER 3 99

A.1 Proof of Theorem 3.3 99
A.2 Proof of Theorem 3.4 101

B. SIMULATION DETAILS FOR CHAPTER 3 103

C. PROOF OF THEOREMS FROM CHAPTER 4 104

C.1 Proofs for Section 4.5.1 104
C.2 Proofs for Section 4.5.2 105
C.3 Proofs for Section 4.5.3 107

D. PROOFS OF LEMMAS FROM CHAPTER 5 108

D.1 Proof of Lemma 5.1 108

D.1.1 Preliminary Definitions 108
D.1.2 Lemma D.1 108
D.1.3 Lemma D.2 110
D.1.4 Lemma D.3 111

D.2 Proof of Lemma 5.2 111

BIBLIOGRAPHY 113

LIST OF TABLES

Table — Page

3.1 Example Race Track Problems 34

3.2 Summary of Learning Performance on the Small Track for RTDP, ARTDP, and RTQ 37

3.3 Summary of Learning Performance on the Larger Track for RTDP, ARTDP, and RTQ 39

4.1 Notation used in the discussion of the TD(λ) learning rule 49

4.2 Notation used in the discussion of Least Squares algorithms. 53

5.1 Cost comparisons between the centralized and distributed adaptive policy iteration algorithms 94

LIST OF FIGURES

Figure Page

2.1 The basic control cycle 8

2.2 An example of a grid world. 10

2.3 The policy iteration algorithm 16

2.4 The computational model for asynchronous dynamic programming . . 17

2.5 The Q-learning algorithm 19

3.1 Real-Time Heuristic Search 22

3.2 Receding Horizon Control 22

3.3 Learning Real-Time A* 23

3.4 Trial-based Real-Time Dynamic Programming 25

3.5 Trial-based Adaptive Real-Time Dynamic Programming 27

3.6 Trial-based Real-Time Q-learning 29

3.7 Example Race Tracks 31

3.8 Performance of Three Real-Time Learning Algorithms on the Small Track 36

3.9 Performance of RTQ on The Small Track for 5,000 Epochs 39

3.10 Performance of Three Real-Time Learning Algorithms on the Larger Track 40

3.11 Performance of RTQ on The Larger Track for 7,500 Epochs 41

3.12 Iterative Deepening RTDP 43

3.13 A recurrent stochastic shortest path problem 44

3.14 Off-line solution to partial proper policy problems 44

4.1 Trial-based TD(λ) for absorbing Markov chains. 50

4.2 Trial-based LS TD for absorbing Markov chains 57

4.3 LS TD for ergodic Markov chains 57

4.4 Comparison of RLS TD and NTD(λ) on a 5 state ergodic Markov chain 65

5.1 The Q-function based policy iteration algorithm 74

5.2 Illustration of the flexible beam 79

5.3 Performance of the adaptive policy iteration algorithm: policy convergence 80

5.4 Performance of the adaptive policy iteration algorithm: parameter convergence 80

5.5 Performance of the adaptive policy iteration algorithm: regulation performance 81

5.6 The impact of insufficient excitation on the performance of the adaptive policy iteration algorithm. 82

5.7 The impact of a short estimation interval on the performance of the adaptive policy iteration algorithm. 82

5.8 State-space model based adaptive optimal control 83

5.9 An optimizing Q-learning algorithm for LQR: Algorithm A 84

5.10 An optimizing Q-learning algorithm for LQR: Algorithm B 85

5.11 SISO adaptive optimal control: delay 2 88

5.12 SISO adaptive optimal control: delay 3 89

5.13 SISO adaptive optimal control: delay 5 90

5.14 The distributed policy iteration algorithm based on Q-learning 92

5.15 Application of the distributed adaptive policy iteration algorithm to a flexible beam 93

5.16 Performance of the distributed adaptive policy iteration algorithm . . 93

CHAPTER 1

INTRODUCTION

Reinforcement learning algorithms based on the principles of Dynamic Programming (DP) have enjoyed a great deal of recent attention both empirically and theoretically. These algorithms have been described generically as Incremental Dynamic Programming (IDP) algorithms [92]. IDP algorithms are intended for use in situations where the information or computational resources needed by traditional DP algorithms [6, 7, 67] are not available. IDP algorithms attempt to find a globally optimal solution by incrementally improving local constraint satisfaction properties as experience is gained through interaction with the environment. Examples of IDP algorithms are Watkins' Q-learning [92], Jalali and Ferguson's Asynchronous Transient Programming [39], Sutton's DYNA family of architectures [83, 84], and Werbos' Heuristic Dynamic Programming [94,95,97]. One of the most widely known of the IDP algorithms is Sutton's Temporal Differences (TD) algorithm [5, 81, 82]. This class of algorithms is not new, going back at least as far as Samuel's [68, 69] adaptive checkers-playing programs, but the links to DP have only been noted and understood very recently. See, for example, the work of Watkins [92], Sutton, Barto, and Williams [88], Sutton [85–87], Barto, Bradtke, and Singh [4], Singh [77], and Werbos [94, 95, 97], among many others. This dissertation presents research that addresses a number of issues concerning the use of IDP algorithms for on-line adaptive optimal control.

1.1 Several Challenges for Incremental Dynamic Programming

DP theory sets forth exact conditions that are required for the convergence of various basic algorithms. Most problems of interest and most practical implementations of the basic algorithms will fail to meet one or more of the necessary convergence conditions *as they are known from the current understanding of the theory.* The limitations and weak spots of the theory are the places that require further research. This section describes several theoretical and practical challenges for IDP that are addressed by this dissertation, prompted by the desire to extend the theory and practice of IDP algorithms to a larger and more generally useful class of problems.

1.1.1 On-line convergence

Many studies have applied IDP algorithms on-line, beginning with Samuel's [68, 69] adaptive checkers-playing programs. This means that the training information available to the algorithm arises as a result of following some trajectory through the state space, and that the functions being approximated by the algorithm are updated after every state transition. Until recently however, there was no theory to give assurances that any such on-line IDP algorithms would converge, even for the simplest case of a finite-state, finite-action Markov decision problem using a lookup-table function approximator. The theory developed by Sutton [82], Dayan [22], and Dayan and Sejnowski [23] came the closest, but their results only applied to learning the value function for absorbing Markov chains, and updates were allowed only after every sequence of state transitions from a start state to a goal state. The theory developed by Jaakkola, Jordan, and Singh [36,37] and by Tsitsiklis [91], which derives Q-learning as a form of stochastic approximation, now gives assurances that some forms of IDP will converge (under the appropriate conditions) when used on-line.

All of the algorithms developed in this thesis address the question of on-line convergence. Chapter 3 develops several new algorithms as generalizations of Korf's LRTA* [48], and proves that two of them converge to the optimal value function and optimal policy when used on-line. Much of the work presented in Chapter 3 has been previously reported by Barto, Bradtke, and Singh [4]. Chapter 4 describes several new algorithms for learning the value function for a Markov chain, and proves that they converge with probability 1 when used on-line. Chapter 5 develops an adaptive policy iteration algorithm based on Q-learning, and proves that it will converge, under the appropriate conditions, to the optimal policy when used on-line.

1.1.2 Continuous Problems

The current theory of IDP algorithms addresses only finite-state, finite-action Markov decision problems. However, a great deal of heuristic research has been done to adapt IDP algorithms to problems with continuous state and action spaces. In fact, the original conception was for continuous spaces, at least, state spaces. Werbos [57, 94, 95, 97] proposed a number of algorithms designed to operate in continuous domains. Perhaps the most closely related approach from the traditional DP literature is the *differential Dynamic Programming* approach of Jacobson and Mayne [38]. Sofge and White [80] describe a system that learns to improve process control with continuous state and action spaces. Some of the many other applications are described by Anderson [1], Jordan and Jacobs [41], and Lin [51,53]. Some of the algorithms used in this work build system models, while others do not. None of these applications, nor many similar applications that have been described, have a firm theoretical grounding.

We address the issue of adapting an IDP algorithm to a continuous domain in Chapter 5. We present an adaptive policy iteration algorithm based on one version of Q-learning, and prove that it converges to the optimal controller under certain conditions. This is the first work of which we are aware that proves convergence for an IDP algorithm when applied to a continuous domain. We also present algorithms derived through a direct translation of a more popular form of the Q-learningalgorithm (which we call *optimizing Q-learning*) from the discrete domain to a continuous domain, and we show analytically that they can converge to destabilizing controllers, *i.e.*, controllers that drive the system state to infinity. This demonstrates that the application of an IDP algorithm to a situation not covered by theory can fail.

1.1.3 Issues of Representation and Function Approximation

The current theory of IDP algorithms requires the function being approximated (the value function or the Q-function) to be stored either in a lookup-table or by a linear function approximator. In either case, there are as many parameters as there are values to represent ($|\text{state space}|$ parameters for the value function, $|\text{state space}| \times |\text{action space}|$ parameters for the Q-function). This means that for many problems of interest it will be impossible to store the function. It is highly desirable to be able to use a more compact function approximation technique in conjunction with an IDP algorithm. By a *compact* function approximator, we mean that the function approximator has fewer independent parameters than there are function input–output mappings to store. The function approximator must perform some sort of generalization in order to store all of the function. As indicated by much heuristic work (Tesauro's TD-Gammon system [89, 90], for example), it *is* possible to achieve good results using a compact function approximator, but a theoretical understanding is still lacking.

There has been some theoretical work that touches on this problem. Singh and Yee [78], Williams and Baird [102], and Bertsekas [6] all consider the problem of using a noisy approximation to a value function as part of a DP algorithm. They bound the error that will result as a function of the magnitude of the approximation noise. But this addresses only part of the problem of using a compact function approximation technique in conjunction with an IDP algorithm. In their studies it has been assumed that the algorithm was able to converge to the approximation in the first place. They do not show that any algorithm will be able to converge to an approximation of the value function when using a compact function approximator.

There have been many *experimental* demonstrations that compact function approximators may be used in conjunction with an IDP algorithm to achieve some level of performance improvement. For example, Tesauro [89, 90] describes a system that learns to play championship level backgammon (which can be viewed as a Markovian decision task) entirely through self-play. It uses a multilayer perceptron trained via backpropagation as a function approximator.

Chapter 5 addresses the issue of convergence when using a compact function approximator in conjunction with an IDP algorithm. As mentioned above, Chapter 5 develops an adaptive policy iteration algorithm based on Q-learning, and applies it to the problem of Linear Quadratic Regulation. Although the Q-function for a Linear Quadratic problem is linear in the Q-function parameters, it is a non-linear (quadratic) function of the input state and action vectors. This is the first instance of which we are aware where it was possible to prove convergence for an IDP algorithm using a compact function approximator that is non-linear in the inputs. Although this does not solve the general problem, it is a first step in that direction.

1.1.4 Model-based vs. Model-free methods

Traditional DP methods are model-based methods. They require the state transition function of the system in order to determine the optimal control law. IDP algorithms such as Q-learning or TD(λ), on the other hand, do not require such a model. Under what conditions would it be advantageous to use a model-free method instead of a model-based method? Gullapalli [31] discusses these questions in detail. We will only present an overview here, under the assumption that no model is initially available. If the model-based approach is chosen, then it will be necessary to build first a model.

There are two primary arguments for taking a model-based approach. First, building a system model and then using that model to solve the optimal control problem is often much easier than trying to solve the optimal control problem directly. Consider, for example, the problem of finding aminimum-time control rule for a linear system. Building a model of a linear system is relatively simple, and derivation of the optimal control law given that model is also a simple matter (see Whittle [100]). However, solving even a very simple problem of this type using a direct or model-free approach similar to that used by Jordan and Jacobs [41] proved extremely difficult [8]. The solution to a minimum–time control problem for a linear system is a bang–bang control rule. The control signal is discontinuous along a "switching curve", taking a maximum value in one direction on one side of the curve and a maximum value in another direction on the other. This is a difficult function to learn using gradient–based techniques.

The second reason to take the model-based approach is that, once the model is formed, it may be used to solve a number of related control problems. For example, the model used to solve one minimum-time control problem may be used to solve many such problems, where the maximum allowed size of the control signal is varied. It may also be used to solve a Linear Quadratic control problem.

The primary argument for taking the model-free approach is that it may be less expensive to find the optimal (or least an acceptable) controller through direct interaction with the system than by building a model and then deriving a controller from the model. Gullapalli [31] describes experiments that support this argument. A model of the state transition function may be relatively easy to build for some dynamic systems, such as linear systems. However, an accurate model may be very difficult to obtain for other systems, such as financial markets, home heating and air conditioning, or flight control. A model for any of these very complex systems will always be idealized, and hence inaccurate, to some degree. Furthermore, even if an accurate model of the system *is* known, it will often be the case that the derivation of an optimal controller directly from that model is analytically and computationally intractable. Consider, for example, the game of backgammon. The state transition function for backgammon is specified by the rules of the game. However, backgammon is estimated to have at least 10^{20} states. Trying to find the optimal policy for the game of backgammon using a traditional model-based approach is an intractable problem.

We address the question of whether or not it is advantageous to use a system model in Chapters 3 and 5. Chapter 3 describes three IDP algorithms, Real-Time Dynamic Programming, Adaptive Real-Time Dynamic Programming, and Real-Time Q-learning, and compares their performance on a benchmark problem. Real-Time Dynamic Programming starts with a model, Adaptive Real-Time Dynamic Programming must build a model through interaction with the system, and Real-Time Q-learning searches for the optimal controller without building a system model. Chapter 3 also describes Iterative Deepening Real-Time Dynamic Programming, and describes situations when it might be preferrable to use the algorithm in simulated interaction with a model of the system. Chapter 5 describes an adaptive policy iteration algorithm based on Q-learning. As applied to the problem of Linear Quadratic Regulation, this algorithm, which does not build a system model, has a computational advantage over the traditional methods for finding the optimal controller for systems meeting a simple constraint.

1.1.5 Hidden state information

The solution to optimal control problems requires the availability of the system state information in some form. However, the system state may not be immediately apparent from the information available at any one state transition. This problem of hidden state information has been termed *perceptual aliasing* [98, 99] by researchers concerned with building autonomous agents that learn through interaction with their environment. The environment for many interesting problems can be modelled as a *Markov decision problem.*

Various approaches to solving the hidden state problem for Markov decision problems have been explored. The simplest of these uses a *tapped delay line representation.* Let x_t be the true state vector at time t, and let the vector y_t be the observation vector at time t. Then the tapped delay line representation for the true state at time t is the concatenation of the last k perceptions y_t through y_{t-k+1} and perhaps the last l control actions u_t through u_{t-l+1}, where k and l are chosen to be long enough to capture the state information. How long is "long enough" depends on the system. A second approach is to build an *observer* that computes an estimate of the true state of the system, $\tilde{x}_t$ as a recursive function of $\tilde{x}_{t-1}$, y_t, and u_{t-1}. The best known observer of this type is the Kalman filter [42, 43]. Lin and Mitchell [54] and Whitehead and Lin [99] discuss the application of both tapped-delay line and observer algorithms to IDP problems with hidden state information. Werbos [94] also discusses incorporating an observer as part of an IDP architecture. Other approaches to the issue of hidden state information in IDP are described by McCallum [59] and Chrisman [17, 18]. Chapman and Kaelbling [13, 14] discuss the related problem of how to extract the relevant state information from a perception y_t that contains much irrelevant information.

We address the problem of hidden state in Chapter 5, where we show how we can use a tapped delay line state representation as input to the adaptive policy iteration algorithm.

1.2 Organization of the Dissertation

Chapter 2 formally defines Markov Decision Problems, and reviews basic DP theory as it applies to the solution of Markov Decision Problems.

Chapter 3 uses the formalism of Markov Decision Problems and the theory of asynchronous DP to develop algorithms that address several of the limitations that are generally encountered in heuristic search algorithms. The newly developed algorithms represent an advance in the theory of heuristic search algorithms by extending Korf's

LRTA* algorithm to stochastic domains, and also an advance in the theory of IDP algorithms.

Chapter 4 reviews a popular IDP algorithm, the TD(λ) learning rule, and considers the use of linear function approximators instead of lookup-tables as part of several alternative learning rules. The chapter describes three new algorithms and proves convergence for their parameter estimates under appropriate conditions. These new algorithms have a number of advantages over previous algorithms. The question of on-line convergence is explicitly addressed.

Chapter 5 develops an adaptive policy iteration algorithm based on Q-learningand applies it to the problem of adaptive Linear Quadratic Regulation. The algorithm is proven to converge to the optimal controller under certain conditions. We illustrate the performance of the algorithm by applying it to a model of a flexible beam. The convergence proof is the first for DP-based reinforcement learning algorithms applied to problems with continuous state and action spaces. The convergence proof is also one of the first *theoretical* results showing that the use of compact function approximators in conjunction with an IDP algorithm can be guaranteed to converge to the optimal. The theory developed in this chapter is a step toward a firm theoretical grounding for problems with continuous state and action spaces.

Chapter 6 concludes and indicates directions for future research.

CHAPTER 2

MARKOV DECISION PROBLEMS AND DYNAMIC PROGRAMMING

We begin this chapter with a formal definition of a Markov Decision Problem (MDP). We give several examples of problems that can be formulated as MDPs in order to promote a more intuitive understanding of this class of problems. The discussion will cover the background necessary to give a basic understanding of MDPs. Additional background needed for the remaining chapters will be provided as necessary. We then review the basic theory of *Dynamic Programming* (DP) as it applies to the solution of MDPs, limiting our discussion to the topics necessary for the development of the dissertation. More comprehensive discussions can be found in Bertsekas [6], Bertsekas and Tsitsiklis [7], and Ross [67].

2.1 Markov Decision Problems

A Markov Decision Problem (MDP) is a stochastic optimal control problem consisting of three components: a discrete-time dynamic system, a one-step, or immediate, cost function, R, and a long-term objective function, V. This review focuses on a particular class of systems: finite-state, finite-action controlled Markov chains. The solution to an MDP is the identification of a control rule, or policy, that optimizes the long-term objective function, given the constraints imposed by the system and the one-step cost function. The meaning of "optimize" will vary depending on the definition of the MDP. Sometimes the optimizing control rule will minimize costs, as in the grid world example (Section 2.1.1). Other times, the optimizing control rule will maximize rewards, as in the backgammon example (Section 2.1.2). In situations when it is desirable to maximize, the one-step cost function is often referred to as the *reward*, or *payoff* function. Except for the backgammon example, all of the problems considered in this dissertation will require cost minimization.

A controlled Markov chain is a discrete-time, stochastic dynamic system with state set, $\mathcal{X}$, action set, $\mathcal{A}$, and a state transition probability function, P. Executing action a from state x causes a transition to state y with probability $P(x, y, a)$. The transition probabilities satisfy the *Markov* property,

$$\begin{aligned} P(x, y, a) &= \text{Prob}\{x_{t+1} = y \mid x_0, a_0, x_1, a_1, \ldots, x_t = x, a_t = a\} \\ &= \text{Prob}\{x_{t+1} = y \mid x_t = x, a_t = a\}. \end{aligned}$$

For each state $x \in \mathcal{X}$ there is a subset, $\mathcal{A}(x) \subseteq \mathcal{A}$, of *admissible* actions from which a must be chosen. Though we assume that the state and action sets are finite, this restriction is not necessary in general. Figure 2.1 illustrates the interaction between the controller and the system. At each time step, t, the controller selects an action, a_t, based on observation of the current state, x_t. The system executes a_t, which results in a state transition to x_{t+1}. The controller also incurs a one-step cost, $r_t = R(x_t, x_{t+1}, a_t)$. In order to simplify the descriptions of DP techniques that will be given below, let us impose an arbitrary, but fixed, ordering on the set $\mathcal{X}$. Then the expression $x < y$ means that the state x comes before the state y in the ordering.

Figure 2.1 implies that the controller directly observes the states of the system. But "state" is an abstract concept. The controller actually observes a *representation* of a state. There may be many ways to represent the states of a given system. For example, a state need not be represented as an indivisible token. The observation $\tilde{x}_t$ is a *valid* representation of the state x_t iff it satisfies the Markov property. That is, knowledge of P, of $\tilde{x}_t$, and of a_t is sufficient to make the best possible prediction of $\tilde{x}_{t+1}$, the representation of state x_t. There is no additional knowledge, including knowledge of previous observations $\{x_i \mid i < t\}$, that could be used to make improved predictions. *We assume throughout this thesis that the controller observes a valid state representation.* Therefore, we will use x_t to represent both the abstract internal state of the system and the observation the controller makes of that state.

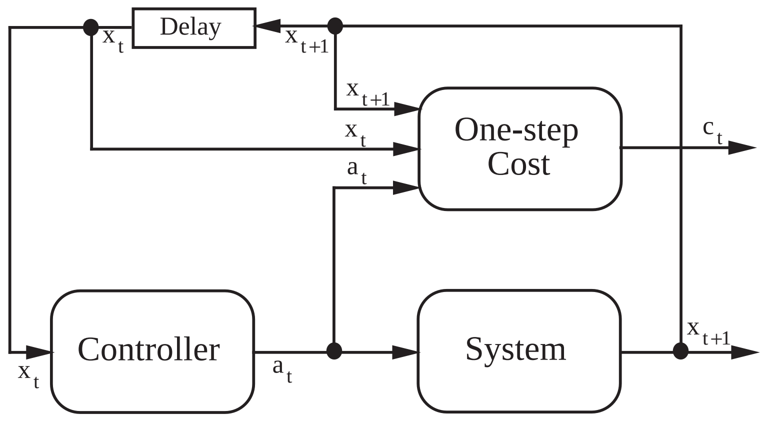


Figure 2.1 The basic control cycle. At each time step, t, the controller selects an action, a_t, based on observation of the current state, x_t. The system executes a_t, which results in a state transition to x_{t+1}. The controller also incurs a one-step cost, r_t.

A control rule, or *policy*, is a function $U\colon \mathcal{X} \times \mathbb{N} \mapsto \mathcal{A}$. U takes a state and a time as its input, and produces the action to be executed at that time as its output: if $x_t = x$, then $a_t = U(x, t)$. A *stationary* policy is one that does not depend on time: $U(x, t_1) = U(x, t_2)$ for all t_1 and t_2. Overloading the notation slightly, we use $a_t = U(x_t)$ to indicate that a_t was selected by the stationary policy U. A policy that depends on state (and possibly time) is referred to as a *closed-loop* policy. An *open-loop* policy is one that depends on time, but not state. An open-loop policy is merely a sequence of actions to be performed one after the other, without reference to the state of the system. The planned state transformation sequences generated by many heuristic search algorithms are open-loop policies. This will be discussed in Section 3.1. A *proper* policy is defined for the special class of MDPs for which there is a designated *goal* state. A policy U is *proper* iff for every state $x \in \mathcal{X}$, taking action $U(x)$ will eventually lead to the goal state with probability one. Unless otherwise noted, we will use the word *policy* to mean *closed-loop policy.*

As stated above, the solution of an MDP is the identification of a policy that optimizes the long-term objective function, V. The long-term objective function is also referred to as the *value* function, the *long-term cost* function, or merely the *cost* function. V is a function of state and policy. There are many possible definitions for V (see, for example, Larson and Casti [50]). We will confine our attention to one of the most common, the *infinite-horizon discounted sum of costs.* This is defined for the policy U and state x_t as

$$V_{\mathrm{U}}(x_t) = \mathcal{E}_{\mathrm{U}}\left\{\textstyle\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right\}. \tag{2.1}$$

This is the expected value of the discounted infinite sum of all costs that will be incurred from the present time onwards, under the assumption that the policy U will be used to choose actions at all times. γ is the discount factor[1]; $0 \leq \gamma \leq 1$. γ can be thought of as an indication of the importance of future events on the value of the present state. If γ is near zero, then future events are not very important for determining the value of the current state. The immediate costs of actions become relatively more important. If γ is near one, then events that may occur in the far future have a relatively greater impact on the value of the current state. If $\gamma < 1$, the *discounted* case, $V_{\mathrm{U}}(x)$ is finite as long as R is finite. If $\gamma = 1$, the *undiscounted* case, then all events, no matter how far into the future, have equal weight. $V_{\mathrm{U}}(x)$ may not be finite in the undiscounted case, depending on the definitions of P and R.

U is an *optimal* policy if $V_{\mathrm{U}}(x) \leq V_{\mathrm{U}_i}(x)$ for all $x \in \mathcal{X}$ and for all policies U_i. There may be more than one optimal policy, but all optimal policies define the same value function, V^*. To give a trivial example, if the one-step cost function, R, is identically zero, then *every* policy defines the same value function (also identically zero), and *every* policy is optimal. Every finite-state, finite-action MDP has an optimal stationary policy under the value function definition given in Equation (2.1), therefore we will be concerned henceforth only with stationary policies.

The following subsections give examples of MDPs that have been used in the study of reinforcement learning algorithms.

2.1.1 The grid world

Figure 2.2 illustrates an example of a simple type of MDP known as a grid world. Each blank cell on the grid is a state of the controlled Markov chain. The shaded cells represent barriers; these are *not* system states. The cell marked with **G** is the goal

[1]The value function as defined in Equation (2.1) clearly depends not only on U and x_t, but also on the value of γ. We omit reference to γ in the notation $V_{\mathrm{U}}(x)$ because γ is usually taken as a parameter that is set as part of the definition of the MDP. The value function cannot be optimized by changing γ.

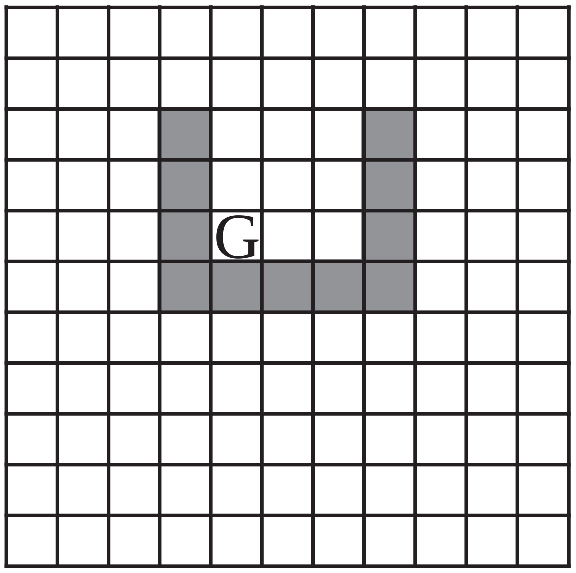


Figure 2.2 An example of a grid world. Each blank cell on the grid represents a state of the controlled Markov chain. The shaded cells represent barriers; these are *not* system states. The cell marked with **G** is the goal state.

state. There are four actions: $\mathcal{A} = \{\mathrm{N}, \mathrm{S}, \mathrm{E}, \mathrm{W}\}$. State transitions are deterministic. Suppose that the system is in state x, and action "N" is chosen. Then the resulting next state, y, is the state directly to the north of x, *if there is such a state.* Otherwise, $y = x$. Similar rules apply for the other three actions. The goal state is a special case. Every action taken from the goal state results in a transition back to the goal state. The cost function is defined by

$$R(x_t, x_{t+1}, a_t) = \begin{cases} 0 & \text{if } x_{t+1} = \mathrm{G} \\ 1 & \text{otherwise.} \end{cases}$$

In effect, the controller is penalized for every time step spent not at the goal. An optimal policy, then, chooses actions so as to minimize the infinite discounted sum of costs, *i.e.*, it chooses actions so as to move into the goal state as quickly as possible.

Grid worlds have been used extensively in the study of reinforcement learning algorithms, for example, by Sutton [85], Moore and Atkeson [62], and Singh [76]. A grid world problem is most easily thought of as an abstraction of a navigation problem. However, a grid world captures many of the essential problems involved in solving more general MDPs, and the intuitive ease of its visual representation has been an important aid in algorithm design and analysis.

2.1.2 Backgammon

Two-person games, such as chess or backgammon, can also be modelled as MDPs. Consider a game of backgammon between players Black and White[2]. We will look at the game from Black's perspective, assuming that White will be following a fixed policy. The first step in modelling backgammon as an MDP is to determine the nature of the controlled Markov chain with which we will be working. What are the state set, the action set, and the state transition rules?

A state consists of a (board pattern, dice roll) pair. It does *not* consist of the board pattern alone. Both the board pattern and the dice roll are needed in order for Black to determine the set of admissible actions. Each action consists of a set of marker movements. Executing an action results in a stochastic state transition, from Black's viewpoint, as follows:

- Black moves his markers in accordance with the chosen action. This step is deterministic, and results in a new board pattern.
- White rolls the dice. This step is stochastic.
- White moves his markers as he chooses, in keeping with his dice roll. This step is deterministic, and results in a new board pattern.
- Black rolls the dice. This step is stochastic.

The game is now in a new state, and Black must again choose an action. The state transition probabilities are determined by the probabilities of the various rolls of the dice and by the policy that White is using to choose his actions. Backgammon has many goal states, *i.e.*, states in which Black wins the game. However, it also has many trap states, *i.e.*, states in which White wins the game. If R gives a value of one for all transitions from a non-goal state into a goal state, and a value of zero for all other transitions, and if $\gamma = 1$, then for any given policy U, $V_{\mathrm{U}}(x)$ will be the probability of eventually reaching a goal state from x while following U. An optimal policy is one with the highest probability of leading to a goal state.

This example shows that MDPs are useful mathematical abstractions that can be used to hide some of the apparent complexity of a problem. States may be complex objects, actions may actually be procedures containing different levels of primitive actions, and state transitions may be accomplished through complex procedures.

2.2 Dynamic Programming

Dynamic programming (DP) as developed to solve MDPs is a set of techniques that can be used to find the value function induced by a given policy, or to find the

[2]We simplify the game to ignore the issues of betting and doubling. The players are only concerned with the fact of winning (or losing) a game. They are not concerned with *degrees* of winning.

optimal policy and the corresponding optimal value function. DP for MDPs is based on the fact that Equation (2.1) can be rewritten as the recurrence relation

$$V_{\mathrm{U}}(x) = \sum_{y \in \mathcal{X}} P(x, y, U(x)) \left[\ R(x, y, U(x)) + \gamma V_{\mathrm{U}}(y) \ \right]. \tag{2.2}$$

This means that the value of state x under policy U is the expected value over all possible next states y of the sum of the immediate return received for executing action $U(x)$ plus the discounted value of y under policy U. Equation (2.2) is true for all policies, including optimal policies. Letting U^* denote an optimal policy, we can rewrite Equation (2.2) for U^* as

$$V_{\mathrm{U}^*}(x) = \sum_{y \in \mathcal{X}} P(x, y, U^*(x)) \left[\ R(x, y, U^*(x)) + \gamma V_{\mathrm{U}^*}(y) \ \right].$$

But all optimal policies, by definition, induce the same value function, V^*. And $V^*(x) \leq V_{\mathrm{U}}(x)$ for all policies U. Therefore the action $a = U^*(x)$ chosen by an optimal policy must minimize the expression

$$\sum_{y \in \mathcal{X}} P(x, y, a) \left[\ R(x, y, a) + \gamma V^*(y) \ \right].$$

Putting all this together, we arrive at Bellman's Optimality Equation

$$V^*(x) = \min_{a \in \mathcal{A}(x)} \left[\sum_{y \in \mathcal{X}} P(x, y, a) \left[\ R(x, y, a) + \gamma V^*(y) \ \right] \right]. \tag{2.3}$$

Once V^* is known, an optimal policy is determined through the equation

$$U^*(x) = \operatorname*{argmin}_{a \in \mathcal{A}(x)} \left[\sum_{y \in \mathcal{X}} P(x, y, a) \left[\ R(x, y, a) + \gamma V^*(y) \ \right] \right]. \tag{2.4}$$

Evaluating the expressions in Equations (2.2) through (2.4) has several drawbacks. First, it requires access to a system model, which may not be available. Second, it requires extensive recomputation if, for some reason, we would want to re-evaluate the expression. Q-functions provide a way around these difficulties. Denardo [26] and Watkins [92] define Q_{U}, the Q-function corresponding to the policy U, as

$$\begin{aligned} Q_{\mathrm{U}}(x, a) &= \sum_{y \in \mathcal{X}} P(x, y, a) \left[\ R(x, y, a) + \gamma V_{\mathrm{U}}(y) \ \right] \\ &= \sum_{y \in \mathcal{X}} \left[\ P(x, y, a) R(x, y, a) \ \right] + \gamma \sum_{y \in \mathcal{X}} \left[\ P(x, y, a) V_{\mathrm{U}}(y) \ \right]. \end{aligned} \tag{2.5}$$

Notice that a can be *any* action in $\mathcal{A}(x)$. It is not necesarily the action $U(x)$ that would be chosen by policy U. The function $Q^* = Q_{\mathrm{U}^*}$ corresponds to the optimal policy. $Q_{\text{U}}(x,u)$ represents the total discounted return that can be expected if any action is taken from state x, and policy U is followed strictly thereafter. We can now rewrite Equations (2.3) and (2.4) using Q-functions. The Bellman Optimality Equation, Equation (2.3), becomes

$$V^*(x) = \min_{a\in\mathcal{A}(x)} \left[\ Q^*(x,a)\ \right], \tag{2.6}$$

and Equation (2.4) becomes

$$U^*(x) = \operatorname*{argmin}_{a\in\mathcal{A}(x)} \left[\ Q^*(x,a)\ \right]. \tag{2.7}$$

Assuming that the function Q^* is stored, then the expressions in Equations (2.6) and (2.7) can be evaluated with much less computation than those in the original Equations (2.3) and (2.4). They can also be evaluated without reference to a system model. The question as to how to acquire knowledge of the function Q^* without first having a system model will be addressed in Section 2.2.4.

2.2.1 Greedy Policies

Any function f that assigns a value to the states of an MDP can be used to define a policy, whether or not $f = V_{\text{U}}$ for any policy U. This policy is called the *greedy* policy with respect to f, and is defined as

$$U^f(x) = \operatorname*{argmin}_{a\in\mathcal{A}(x)} \left[\sum_{y\in\mathcal{X}} P(x,y,a) \left[\ R(x,y,a) + \gamma f(y)\ \right] \right]. \tag{2.8}$$

The policy U^f is also called the *certainty equivalence optimal policy*, since, under the certainty equivalence assumption that f is an accurate model of the optimal evaluation function, $U^f = U^*$.

Any given policy U is the greedy policy defined by an infinite number of functions. This is easy to see. Let $f(x) = V_{\text{U}}(x) + \epsilon(x)$, where U is some policy, and $\epsilon(x)$ is a small perturbation. Then $\epsilon(x)$ can vary about a small neighborhood of zero without changing the value $U^f(x)$ returned by Equation (2.8). In particular, an optimal policy is the greedy policy for many different functions. It is not necessary to know the optimal value function exactly before the optimal policy can be defined.

2.2.2 Synchronous DP

The techniques described in this section build estimates for value functions and optimal policies *synchronously*, in the sense that the estimates are built in stages. The estimate for each state is updated once, and only once, per stage.

2.2.2.1 Evaluating a policy

There are two principle ways to evaluate a given policy, U, *i.e.*, to find V_{U}. The first is a method of successive approximations that converges over a number of iterations to a function that satisfies Equation (2.2). The second method uses matrix algebra to solve for V_{U}. (There are other methods too, of course, such as Monte Carlo approximation.)

The method of successive approximations has itself two main variations. The *Jacobi* method updates the values for all states simultaneously. If we let V_k denote the k^{th} approximation to V_{U}, then the Jacobi method uses the following update formula:

$$V_{k+1}(x) = \sum_{y \in \mathcal{X}} P(x, y, U(x)) \Big[\; R(x, y, U(x)) + \gamma V_k(y) \; \Big]. \tag{2.9}$$

The process of updating the value for state x is known as a *backup*. The *Gauss-Seidel* method updates the values for the states in a series of sweeps. If we define

$$\tilde{V}_k(x, y) = \begin{cases} V_{k+1}(y) & \text{if } y < x \\ V_k(y) & \text{otherwise} \end{cases}, \tag{2.10}$$

then the Gauss-Seidel method sweeps along the state ordering from the "smallest" to the "largest" using the update formula:

$$V_{k+1}(x) = \sum_{y \in \mathcal{X}} P(x, y, U(x)) \Big[\; R(x, y, U(x)) + \gamma \tilde{V}_k(x, y) \; \Big]. \tag{2.11}$$

The Gauss-Seidel method could use a different ordering for every sweep. It is only important that every state be updated once per sweep.

Both the Jacobi and the Gauss-Seidel methods are guaranteed to converge to V_{U}, assuming that V_{U} is finite. If there are states for which $V_{\mathrm{U}}(x)$ is not finite, the estimates for those states will grow without bound. The estimates for the remaining, finite valued states will converge. V_{U} will always be finite in the discounted case. However, in the undiscounted case it is possible that V_{U} takes on infinite values for at least some states. The Gauss-Seidel method typically converges faster because it always uses the latest estimates available when doing an update.

The other method of evaluating the policy U is to simply calculate V_{U} using matrix algebra. Since there are only a finite number of states, we can represent the function V_{U} as a vector. Now, define the vector R_{U} so that

$$[R_{\mathrm{U}}]_x = \sum_{y \in \mathcal{X}} P(x, y, U(x)) R(x, y, U(x)).$$

Next, define the matrix P_{U} so that $[P_{\mathrm{U}}]_{xy} = P(x, y, U(x))$. These definitions allow us to rewrite Equation (2.2) using matrix notation as $V_{\mathrm{U}} = R_{\mathrm{U}} + \gamma P_{\mathrm{U}} V_{\mathrm{U}}$. Simple matrix algebra then yields $V_{\mathrm{U}} = \Big[\; I - \gamma P_{\mathrm{U}} \; \Big]^{-1} R_{\mathrm{U}}$. V_{U} is finite as long as $\Big[\; I - \gamma P_{\mathrm{U}} \; \Big]$ is non-singular.

2.2.2.2 Value Iteration

Value iteration is a procedure for finding the optimal value function without requiring knowledge of an optimal policy beforehand. Value iteration is a method of successive approximations that converges over a number of iterations to a function that satisfies Equation (2.3), just as the policy evaluation procedure converges to a function that satisfies Equation (2.2). As with policy evaluation, both Jacobi and Gauss-Seidel versions of the algorithm are possible. The Jacobi version uses the update rule

$$V_{k+1}(x) = \min_{a \in \mathcal{A}(x)} \left[\sum_{y \in \mathcal{X}} P(x,y,a) \left[\; R(x,y,a) + \gamma V_k(y) \; \right] \right], \tag{2.12}$$

to update the value estimates for all states simultaneously. The Gauss-Seidel version uses the update rule

$$V_{k+1}(x) = \min_{a \in \mathcal{A}(x)} \left[\sum_{y \in \mathcal{X}} P(x,y,a) \left[\; R(x,y,a) + \gamma \tilde{V}_k(x,y) \; \right] \right], \tag{2.13}$$

where $\tilde{V}_k(x,y)$ is as defined in Equation (2.10), to update the value estimates sequentially, always using the most recent available information.

Value iteration is guaranteed to converge to V^* whenever V^* is finite. This will always be true in the discounted case. Restrictions on the cost function or on the form of the controlled Markov chain are necessary in order for V^* to be finite in the undiscounted case. We will discuss this issue in Chapter 3.

2.2.2.3 Policy Iteration

Howard [33] first described the process of policy iteration. Policy iteration explicitly searches for the optimal policy using the algorithm outlined in Figure 2.3. The heart of the algorithm lies in the equation

$$U_{k+1}(x) = \operatorname*{argmin}_{a \in \mathcal{A}(x)} \left[\sum_{y \in \mathcal{X}} P(x,y,a) \left[\; R(x,y,a) + \gamma V_{U_k}(y) \; \right] \right]. \tag{2.14}$$

Howard showed that by using the value function for U_k in this way to define U_{k+1}, we are assured that U_{k+1} will be at least at good as U_k. If we then note that there are only a finite number of stationary policies for a finite MDP, we see that the policy iteration algorithm must terminate with an optimal policy.

Policy iteration requires a number of (policy evaluation, policy improvement) cycles. Each policy evaluation step can be as expensive as the entire value iteration algorithm. This means that policy iteration, though it may converge very rapidly to the optimal policy in terms of the number of these cycles, typically takes more total computation than value iteration.

```
1  $k = 0$
2  Choose an initial policy, $U_1$.
3  **repeat** {
4      Increment $k$.
5      Evaluate policy $U_k$, finding function $V_{U_k}$.
6      Define an improved policy $U_{k+1}$ using Equation (2.14).
7  } **until** $(U_k = U_{k+1})$
```

Figure 2.3 The policy iteration algorithm.

2.2.3 Asynchronous DP

Synchronous DP, as described in Section 2.2.2, takes place in a series of well-defined stages. In *asynchronous* DP, on the other hand, this concept of stages is eliminated. Figure 2.4 illustrates the computational model we will be using for asynchronous DP[3] [7]. Each state x_i is identified with a processor, also labelled x_i. The processors do not have access to a central clock, and they may operate at very different speeds. The processors exchange information via communication links, which can suffer arbitrary (but not infinite) transmission delays. Each processor holds a partial model of the MDP; it "knows" the information relevant to its state. Processor x knows $\mathcal{A}(x)$, $P(x,y,a)$ and $R(x,y,a)$ for all possible successor[4] states y and for all $a \in \mathcal{A}(x)$.

$V_t(x)$ represents the value function estimate for state x at time t, where t is a real number measured externally to any of the processors. We introduce the functions $t(\cdot,\cdot)$ and $\tau(\cdot,\cdot,\cdot)$ as notational aids in description of asynchronous DP. These functions would not actually be used as part of an implementation. The value of $t(x,k)$ is the time at which processor x updates the value estimate for state x for the k^{th} time. The value function estimate changes only at these times, so that

$$V_t(x) = V_{t(x,k)}(x) \qquad \text{for all} \qquad t(x,k) \leq t < t(x,k+1).$$

The value $\tau(x,y,t)$ is the time $t(y,j) \leq t$ of the latest value estimate from processor (and state) y that is available to processor x at time t. The interval $t - \tau(x,y,t)$ is the transmission delay. The asynchronous value iteration backup of state x at time $t = t(x,k+1)$ can be represented in this notation as

$$V_t(x) = \min_{a \in \mathcal{A}(x)} \left[\textstyle\sum_{y \in \mathcal{X}} P(x,y,a) \left[R(x,y,a) + \gamma V_{\tau(x,y,t)}(y) \right] \right]. \tag{2.15}$$

This equation is a generalization of the synchronous value iteration equations (Equation (2.12) and Equation (2.13)). The notion of stages has been eliminated. Backups

[3]We will actually be considering the situation that Bertsekas and Tsitsiklis [7] call *totally* asynchronous DP.

[4]The state x is a *successor* of state y iff there exists an $a \in \mathcal{A}(y)$ such that $P(y,x,a) \neq 0$. The state x is a *predecessor* of state y iff there exists an $a \in \mathcal{A}(x)$ such that $P(x,y,a) \neq 0$.

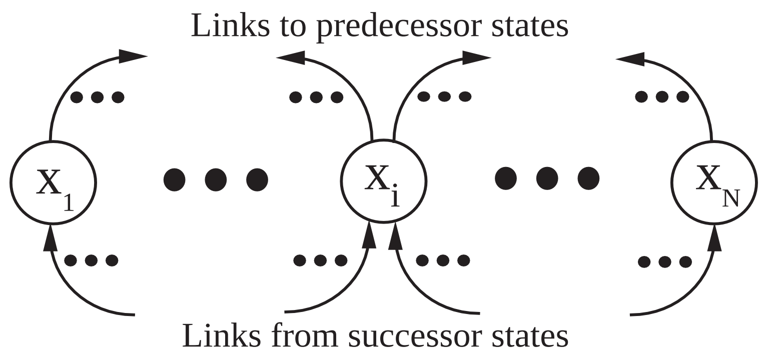


Figure 2.4 The computational model for asynchronous dynamic programming. The labelled circles represent the nodes of the computation network. Each node is identified with a state of the controlled Markov chain. The communication links transfer value function estimates from each state back to the predecessor states.

can occur in an arbitrary order, and at arbitrary rates. It is possible for the value estimate for one state to be backed up many times before the estimate for another state is backed up even once. This is a potential waste of computational resources, but it is a result of the very limited requirements placed on the communication between the processors. The analogous asynchronous generalization of the synchronous policy evaluation equations (Equation (2.9) and Equation (2.11)) is

$$V_t(x) = \sum_{y \in \mathcal{X}} P(x, y, U(x)) \left[\; R(x, y, U(x)) + \gamma V_{\mathcal{T}(x,y,t)}(y) \; \right]. \tag{2.16}$$

An asynchronous policy update is given by the equation

$$U_t(x) = \operatorname*{argmin}_{a \in \mathcal{A}(x)} \left[\; \textstyle\sum_{y \in \mathcal{X}} P(x, y, a) \left[\; R(x, y, a) + \gamma V_{\mathcal{T}(x,y,t)}(y) \; \right] \; \right]. \tag{2.17}$$

Theorem 2.1, which is a summarization of several theorems due to Bertsekas and Tsitsiklis [7], describes conditions under which asynchronous value iteration and policy evaluation are guaranteed to converge to V^* or V_U respectively. Theorem 2.1 applies to the discounted case, that is, whenever $\gamma < 1$. The convergence conditions are more complicated in the undiscounted case, and will be discussed in chapter 3.

Theorem 2.1 *Asynchronous value iteration and policy evaluation are guaranteed to converge to V^* or V_U respectively, when applied to a discounted MDP, whenever the following conditions are satisfied: 1) the value estimates for all states are updated infinitely often; 2) $\lim_{k \to \infty} t(x, k) = \infty$, for all $x \in \mathcal{X}$; and 3) transmission delays, while unbounded, are not infinite.*

Asynchronous policy iteration requires that value function updates (Equation (2.15) and Equation (2.16)) be interspersed somehow with the policy updates described in Equation (2.17). Using policy updates alone will only achieve one policy improvement step as described in Section 2.2.2.3. Williams and Baird [101]) discuss asynchronous policy iteration in detail, and give a number of conditions guaranteeing convergence.

2.2.4 Temporal Difference Methods

The DP algorithms described in Sections 2.2.2 and 2.2.3 are all model-based algorithms. They require a model of the controlled Markov chain: the transition probability function P and of the one-step cost function R. But a system model is not always available. Temporal difference (TD) methods [82] are model-free methods designed to solve MDPs. TD methods grew out of a study of reinforcement learning algorithms and the problem of temporal credit assignment [1,5,81]. It was only some time after development that the relationship between TD methods and asynchronous dynamic programming was realized [4,85,92]. It is now commonly understood [37, 77,91] that TD methods use a form of asynchronous stochastic approximation to find a solution to recurrence relations like those descibed in Equation (2.2) and Equation (2.3). The two most important TD methods are the TD(λ) and the Q-learning algorithms. We will only discuss the Q-learning algorithm in this chapter, deferring discussion of TD(λ) to Chapter 4.

Q-learning, first described by Watkins [92], uses stochastic approximation to build an estimate for the function Q^*. Lukes, Thompson, and Werbos [57] also describe a version of the algorithm. The Q-learning algorithm can be derived by starting with the observation that the combination of Equation (2.2) and Equation (2.5) yields

$$\begin{aligned} Q_{\mathrm{U}}(x, U(x)) &= \sum_{y \in \mathcal{X}} P(x, y, U(x)) \Big[\ R(x, y, U(x)) + \gamma V_{\mathrm{U}}(y)\ \Big] \\ &= V_{\mathrm{U}}(x). \end{aligned} \tag{2.18}$$

Now, Equation (2.18) can be combined once again with Equation (2.5) to give

$$Q_{\mathrm{U}}(x, a) = \sum_{y \in \mathcal{X}} \Big[\ P(x, y, a) R(x, y, a)\ \Big] + \gamma \sum_{y \in \mathcal{X}} \Big[\ P(x, y, a) Q_{\mathrm{U}}(y, U(y))\ \Big]. \tag{2.19}$$

The function Q_{U} has been described as a recurrence relation, without reference to V_{U}. A Bellman-style optimality equation for Q-functions is similarly derived,

$$\begin{aligned} Q^*(x, a) &= \sum_{y \in \mathcal{X}} \Big[\ P(x, y, a) R(x, y, a)\ \Big] + \gamma \sum_{y \in \mathcal{X}} \Big[\ P(x, y, a) Q^*(y, U^*(y))\ \Big] \\ &= \sum_{y \in \mathcal{X}} \Big[\ P(x, y, a) R(x, y, a)\ \Big] + \gamma \sum_{y \in \mathcal{X}} \left[P(x, y, a) \min_{b \in \mathcal{A}(y)} Q^*(y, b) \right] \end{aligned} \tag{2.20}$$

Equations (2.19) and (2.20) lead directly to the two forms of the Q-learning algorithm. *Policy-based* Q-learning builds an estimate of the function Q_{U} for a fixed policy U. *Optimizing* Q-learning builds an estimate of the optimal Q-function, Q^*. Like TD(λ), Q-learning builds its function estimates based on information *sampled* from the environment during actual state transitions. The policy-based Q-learning rule is

$$Q_{k+1}(x_k, a_k) = Q_k(x_k, a_k) +$$

$$\alpha_k(x_k, a_k) \Big[\; R(x_k, y_k, a_k) + \gamma Q_k(y_k, U(y_k)) - Q_k(x_k, a_k) \; \Big], \qquad (2.21)$$

where Q_k represents the estimate of Q_{U} at time k, and $\alpha_k(x_k, a_k)$ is the learning rate at time k for the (state, action) pair (x_k, a_k). The optimizing Q-learning rule is

$$\begin{aligned} Q_{k+1}(x_k, a_k) = {} & Q_k(x_k, a_k) + \\ & \alpha_k(x_k, a_k) \left[R(x_k, y_k, a_k) + \gamma \min_{b \in \mathcal{A}(y_k)} [Q_k(y_k, b)] - Q_k(x_k, a_k) \right], \end{aligned} \qquad (2.22)$$

where Q_k now represents the estimate of Q^* at time k. Figure 2.5 summarizes the Q-learning algorithm.

```
1   for $k = 1$ to $\infty$ {
2       Choose a state $x_k$ from $\mathcal{X}$.
3       Choose an action $a_k$ from $\mathcal{A}(x_k)$.
4       Execute action $a_k$ from state $x_k$. This results in a
        stochastic transition to state $y_k$, and incurs the cost
        $r_k = R(x_k, y_k, a_k)$.
5       Update the estimate $Q_k$ using the update rule given in
        Equation (2.21) in order to learn $Q_{\mathrm{U}}$, or using the update
        rule given in Equation (2.22) in order to learn $Q^*$.
6   }
```

Figure 2.5 The Q-learning algorithm.

Convergence of the optimizing Q-learning algorithm has been shown using a number of techniques [37, 91–93]. Each of these proofs shows that Q_k is guaranteed to converge asymptotically to Q^* under the assumption that every admissible action is chosen from every state an infinite number of times, and that the learning rates, $\alpha_k(x_k, a_k)$, are asymptotically reduced to zero in an appropriate way, as $k \to \infty$. Perhaps the most general convergence proof is that of Tsitsiklis [91], which we restate as follows:

Theorem 2.2 (Tsitsiklis) *If the learning rates $\alpha_k(x, a)$ are random numbers such that (1) $\sum_{k=1}^{\infty} \alpha_k(x, a) = \infty$, with probability one, and (2) $\sum_{k=1}^{\infty} \alpha_k(x, a)^2 < \infty$, with probability one, then $Q_k(x, a)$ updated using the optimizing Q-learning rule converges to $Q^*(x, a)$, with probability one, for every state x and action a, in each of the following cases: (a) if $\gamma < 1$; (b) if $\gamma = 1$, $Q_0(x, a) = 0$ for every absorbing state x, and all policies are proper; or (c) if $\gamma = 1$, there exists at least one stationary proper policy, every improper stationary policy yields infinite expected cost for at least one state, $Q_0(x, a) = 0$ for every absorbing state x, and Q_k is guaranteed to be bounded, with probability one.*

CHAPTER 3

REAL-TIME DYNAMIC PROGRAMMING

State space search lies at the heart of many of the systems produced by Artificial Intelligence. However, most of the algorithms designed to perform this search suffer from several limitations:

1. they do not take advantage of problem solving experience to improve performance on subsequent problems;

2. they are limited to deterministic environments;

3. they require an exact model of the problem; and

4. they do not act in real time.

It is the goal of a great deal of current research to relax these limitations, or to eliminate them altogether. See, for example, [19, 35, 48, 63, 71–73]. In this chapter we use the formalism of Markov decision problems and the theory of asynchronous DP to address each of these limitations. Much of the work presented here has been previously reported by Barto, Bradtke, and Singh [4].

This chapter is organized as follows. Section 3.1 discusses heuristic search and the recent work that has been done to address the limitations listed above. Section 3.3 describes the class of *stochastic shortest path problems*, which can be viewed as an MDP formalization of the sorts of problems encounted by heuristic search. Section 3.4 presents the Real-Time Dynamic Programming (RTDP) algorithm, and Section 3.5 describes an adaptive version, Adaptive RTDP. Section 3.6 goes on to describe an on-line version of Q-learning, which will be used for comparison purposes in a series of experiments. The experiments are described in Section 3.8. Section 3.9 proposes a generalization of RTDP that is applicable to a wider class of problems. Finally, Section 3.10 gives concluding remarks.

3.1 State–Space Search

A state-space search problem is defined by a set of problem states, a set of operators that map states to states, a set of initial state, and a set of goal states. The objective is to find a sequence of operators, a *plan*, that maps the initial state to one of the goal states and (possibly) optimizes some measure of cost, or merit, of the solution path. Problems of this type can be considered as *shortest path* problems. The search algorithm is trying to find the shortest (least costly, or optimal) path from the selected start state to the set of goal states. Shortest path problems are a form of MDP. The plan generated by the search algorithm corresponds to an open-loop policy as described in Chapter 2.

State-space search algorithms typically do not learn from one problem solving episode to the next, even though each of the problems may have common components. The algorithm must start from scratch for each new problem. We are aware of only a few exceptions to this generalization. Mérõ [60], Gelperin [28], and Korf [48] each propose and prove the convergence of heuristic state-space search algorithms that modify the heuristic evaluation function (h) from one problem solving trial to the next based on experience[1]. The adaptive heuristic game-tree search algorithms Samuel developed for his checkers-playing programs [68, 69] are similar, but no convergence proofs are available. Although these algorithms use DP-like backups to update the heuristic evaluation functions, they were developed independently of DP. Korf's Learning-Real-Time-A* (LRTA*) algorithm is the most relevant to the work described in this chapter.

In many applications of state-space search algorithms, the generation of a problem-solving plan is not the final objective. The intent may be to execute the plan (operator sequence) to generate a time sequence of actual inputs to a physical system. A plan executed in this manner, without reference to the state transitions actually induced, will only produce satisfactory results when the state transitions are deterministic and when the problem model used by the search algorithm is accurate and complete. Recent work on the development of "universal plans" [72] seeks to address this limitation. See, for example, Chapman [12], Ginsberg [29], and Schoppers [73]. The idea of a universal plan corresponds to the idea of a closed-loop policy as described in Chapter 2.

Real-time heuristic search algorithms, as discussed by Korf [48], are required to perform interactively with the problem environment. Instead of separate planning and execution phases, planning (through state-space search) and execution must be interleaved as described in Figure 3.1. There are a number of reasons that would prompt this interleaving. First of all, the existance of hard time constraints in the problem environment, in conjuction with the size of the problem state space, may preclude the ability to run a search to completion before time limits are reached. Obviously, this reason becomes less important when the state space is small, but for many problems, such as chess or natural language understanding, the size of the state space implies that no search could be completed before any reasonable time limits were reached. Second, unanticipated state transitions may invalidate any plan. As described above, open-loop plans can fail when faced with a stochastic or incompletely modelled environment. In contrast to traditional heuristic search algorithms, since real-time heuristic search selects an action at every time step based on the current state of the system, it can be viewed as a (possibly nonstationary) closed-loop control policy. This method of generating a closed-loop policy as the result of the iterative redesign and execution of open-loop policies (plans) is known as *receding horizon*

[1]Heuristic search algorithms attempt to improve performance by guiding the search process in some manner through the use of domain knowledge. See, for example, Nilsson [64], Charniak and McDermott [15], Winston [103], and Barr and Feigenbaum [3].

control by control engineers [49, 58]. Figure 3.2 describes receding horizon control. It is readily seen through a comparison of Figures 3.1 and 3.2 that real-time heuristic search is a form of receding horizon control.

```
1  repeat until stopping conditions satisfied {
2      Design a plan based on the current state. The design/search process
       continues until time constraints force termination, or until the best possible
       answer is found.
3      Execute the action selected. This causes a transition to a new state, one
       possibly unanticipated by the planning step.
4  }
```

Figure 3.1 Real-Time Heuristic Search. The basic execution cycle for real-time heuristic search as discussed by Korf [48].

```
1  repeat until stopping conditions satisfied {
2      Design a controller based on current knowledge of the system.
3      Execute the policy defined by the controller.
4  }
```

Figure 3.2 Receding Horizon Control. A receding horizon control procedure cycles between design and execution phases. The design phase continues until time constraints force termination, or until a suitable answer is derived. The execution phase may continue for an arbitrary length of time, but will terminate when conditions indicate that redesign of the controller is necessary.

3.2 Learning Real-Time A*

Korf's LRTA* [48] is a learning real-time heuristic search algorithm. Figure 3.3 gives an outline of the algorithm. The quantity $c(n_1, n_2)$ is the transition cost to go from node (problem state) n_1 to its neighbor n_2. The function $\tilde{h}$ is the modifiable heuristic evalation function. If node n *has not* been visited before, then $\tilde{h}(n)$ is $h(n)$, the initial heuristic value, possibly augmented by a look-ahead search. If node n *has* been visited previously, then $\tilde{h}(n)$ is the value stored for node n at that last visit. Line 4 of Figure 3.3 changes the heuristic value of the current state to be the minimum over all neighbors of the cost to move to that neighbor, plus the estimated cost to go from that neighbor to the goal. Line 5 chooses the next state to be one of those that minimizes the total estimated cost to go. Ties are broken randomly. Korf [48] proves the following convergence theorem:

Theorem 3.1 (Korf) *In a finite problem space with positive edge costs, and non-overestimating initial heuristic values, in which a goal state is reachable from every state, over repeated trials of LRTA*, the heuristic values will eventually converge to their exact values along every optimal path.*

```
Set $k = 0$.
Select the start state, $s_0$.
while not Goal($s_k$) {
    $\tilde{h}(s_k) = \min_{n \in \texttt{Neighbors}(s_k)} \left[ c(s_k, n) + \tilde{h}(n) \right]$
    $s_{k+1} = \operatorname{argmin}_{n \in \texttt{Neighbors}(s_k)} \left[ c(s_k, n) + \tilde{h}(n) \right]$
    $k = k + 1$
}
```

Figure 3.3 Learning Real-Time A*.

LRTA* can be implemented as a real-time system because each step can be performed in at most time $O(N)$, where N is the total number of nodes, or states, in the problem. It could be much better than this, as the cost actually depends on the maximum branching factor. Comparing line 4 to Equation (2.15), and line 5 to Equation (2.17), we see that LRTA* implements an asynchronous policy iteration algorithm on undiscounted, deterministic shortest path problems. LRTA* implements policy iteration because it changes its policy, the choice of action to take, as it updates the value estimates of the available estimates. It is asynchronous policy iteration since the policy change is made only at the current state. Although not explicitly designed as such originally, LRTA* is the result of interleaving the steps of DP with the actual process of control so that control policy design occurs concurrently with control.

3.3 Stochastic Shortest Path Problems

A *stochastic shortest path problem* (also known as a *first passage* problem [7]) is a generalization of the shortest path problem. A stochastic shortest path problem is an MDP with a set of *start* states, $S \subseteq \mathcal{X}$, and a set of goal states, $G \subseteq \mathcal{X}$. The goal states are absorbing, so that any action taken from a goal state results in a transition to another goal state. The cost $R(g_1, g_2, a) = 0$ for any two goal states g_1 and g_2, and for any admissible action a. Therefore, no further costs are incurred once the system has entered the goal set. This means that the value function V_U is finite for any policy U that takes the system into the goal set, even in the undiscounted case (when $\gamma = 1$). This is true because only a finite number of the immediate costs incurred by following U to the goal set may be non-zero. U is a *proper* policy if following U from any $x \in \mathcal{X}$ will eventually lead, with probability 1, to the goal set. U is an *improper* policy if following U generates an ergodic set of states that does not intersect the goal set. It is impossible to get from this ergodic set to the goal set while following U.

The solution to a deterministic shortest path problem is a policy that produces the shortest paths from the start states to the goal states. The solution to a stochastic shortest path problem is a policy that chooses actions that minimize the *expected* cost to reach the goal states.

The value iteration and policy evaluation algorithms discussed in Chapter 2 are guaranteed to converge for any discounted MDP. This holds for both the synchronous and asynchronous algorithms. Convergence requires further assumptions in the undiscounted case. Theorem 3.2 describes the conditions under which asynchronous value iteration will converge when applied to undiscounted stochastic shortest path problems.

Theorem 3.2 (Bertsekas and Tsitsiklis) *Asynchronous value iteration will converge to V^* when applied to an undiscounted stochastic shortest path problem when: 1) there exists at least one proper stationary policy; 2) every improper stationary policy yields infinite cost for at least one state; and 3) the conditions of Theorem 2.1 are satisfied.*

The second assumption means that if the non-goal state x is in an ergodic set generated by the improper policy U, then

$$\begin{aligned} V_{\mathrm{U}}(x) &= \mathcal{E}_{\mathrm{U}}\left\{\textstyle\sum_{i=0}^{\infty} \gamma^i R(x_i, x_{i+1}, U(x_i)) \mid x_0 = x\right\} \\ &= \mathcal{E}_{\mathrm{U}}\left\{\textstyle\sum_{i=0}^{\infty} R(x_i, x_{i+1}, U(x_i)) \mid x_0 = x\right\} \\ &= \infty. \end{aligned}$$

Some of the immediate costs may be positive, some negative, and some zero, but the total expected value must be infinite.

3.4 Real-time Dynamic Programming

Trial-based Real-Time Dynamic Programming (RTDP) is the generalization of Korf's LRTA* to handle stochastic shortest path problems[2]. As such, it is a learning receding horizon control procedure. Figure 3.4 gives a pseudocode specification for RTDP. We use the subscript k to count the total number of design/execution phases. During the k^{th} design phase, trial-based RTDP will backup the value of all states x in some set $B_k \subseteq \mathcal{X}$ (line 5). The current state, x_k, is always a member of B_k. It is possible that the value estimates for some states in B_k will be updated many times during a given step. After updating the value function, RTDP chooses the action to take from the current state in a best-first, or greedy fashion (line 6).

We need the notion of a *relevant* state before we talk about the convergence of RTDP. A state is relevant to the solution of a stochastic shortest path problem if it is a start state, or can be reached from the start states when following an optimal policy. It is possible that all of the states are relevant. In a deterministic problem, the relevant states are only those lying on the shortest paths from the start set to the

[2]Ishida and Korf [35] and Ishida [34] have previously designed a generalization of LRTA*, called *Moving Target Search*, for a special class of stochastic shortest path problems.

```
1  Set $k = 0$.
2  **repeat** forever {
3      Set $x_k$ to a randomly selected start state.
4      **while** ($x_k$ is not a goal state) {
5          Perform the backup operation.
           $$V_{k+1}(x) = \begin{cases} \min_{a \in \mathcal{A}(x)} \sum_{y \in \mathcal{X}} P(x,y,a) \Big[ \; R(x,y,a) + \gamma V_k(y) \; \Big] & \text{if } x \in B_k \\ V_k(x) & \text{otherwise} \end{cases}$$
6          Select the best action, given the *current estimate* of the optimal value
           function.
           $$a_k = \operatorname*{argmin}_{a \in \mathcal{A}(x_k)} \sum_{y \in \mathcal{X}} P(x_k,y,a) \Big[ \; R(x_k,y,a) + \gamma V_{k+1}(y) \; \Big]$$
7          Perform the action and get the next state, $x_{k+1}$.
8          $k = k + 1$.
9      }
10 }
```

Figure 3.4 Trial-based Real-Time Dynamic Programming. k counts the total number of design/execution phases since the algorithm was started.

goal set. For example, if we choose a particular cell of the grid world (Section 2.1.1) as the start state, then nearly all of the system states are irrelevant.

Asynchronous policy iteration (lines 5 and 6) is RTDP's heart. Using the theory of asynchronous DP as descibed by Bertsekas and Tsitsiklis [7] we present the following convergence theorem for RTDP. The proof is given in Appendix A.1. In order to use the theory developed by Bertsekas and Tsitsiklis, we assume that the immediate costs depend only on the current state and action, *i.e.*, they are of the form $R(x, a)$. It is possible to extend the basic theory to cover the more general form $R(x, y, a)$, but that is not necessary for the purposes of this discussion. RTDP will converge to a stationary optimal policy if ties in line 6 are resolved deterministically, or when there are no ties to be broken. RTDP will converge to a nonstationary optimal policy otherwise.

Theorem 3.3 (Trial-based RTDP) *In undiscounted stochastic shortest path problems, Trial-Based RTDP, with the initial state of each trial restricted to a set of start states, converges (with probability one) to V^* on the set of relevant states, and the controller's policy converges to an optimal policy (possibly nonstationary) on the set of relevant states, under the following conditions: 1) there is at least one proper stationary policy; 2) all immediate costs incurred by transitions from non-goal states are positive, i.e., $R(x, a) > 0$ for all non-goal states x and for all $a \in \mathcal{A}(x)$; and 3) the initial value estimates of all states are non-overestimating, i.e., $V_0(x) \leq V^*(x)$ for all states $x \in \mathcal{X}$.*

The requirement in Theorem 3.3 that $R(x,a) > 0$ for all non-goal states x is somewhat stronger than the requirement in Theorem 3.2 that every improper stationary policy yields infinite cost for at least one state. This stronger condition is necessary because of the on-line nature of trial-based RTDP. The value estimates of all of the states must be updated infinitely often in order to ensure convergence of the asynchronous DP algorithms. However, the manner in which states are updated by RTDP is determined by the state trajectory encountered during interaction with the system. Even though the value estimates for more than one state may be updated at each time step, the trajectory may be such that some states are never updated, or are only updated a finite number of times. However, the only states we really care about are the relevant states, and Theorem 3.3 assures us that value estimates and the policy for these states will converge to the optimal.

The on-line nature of RTDP can actually confer a great advantage in terms of *total* computation required. As the algorithm progresses, it focusses its computational resources more and more narrowly upon the relevant states. It learns to ignore those states that can not be reached by an optimal policy. This effect will be demonstrated experimentally in Section 3.8.

3.5 Adaptive Real-time Dynamic Programming

RTDP requires a model of the system. It needs to know $P(x,y,a)$ and $R(x,y,a)$ for all x, y, and a. This information may not always be available. One way around this problem, of course, would be to build a model of the system first, and then to apply RTDP. Another approach would be to merge the model building process with RTDP. The Adaptive RTDP (ARTDP) algorithm described in Figure 3.5 takes the latter approach. It uses counters to build a maximum likelihood model of the state transition probabilities based on the transitions experienced as it interacts with the system. The value $\eta(x,y,a)$ is the number of experienced state transitions from state x to state y upon application of the action a. The value $\eta(x,y)$ is the number of experienced state transitions from state x to state y upon application of *any* action, *i.e.*, $\eta(x,y) = \sum_{a \in \mathcal{A}(x)} \eta(x,y,a)$. The maximum likelihood estimate for the transition probability $P(x,y,a)$ is $\hat{P}(x,y,a) = \eta(x,y,a)/\eta(x,y)$. As described here, ARTDP must build a model of the state transition probabilities, but it already knows the immediate cost function, R. This is not an unrealistic assumption for many problems, where the immediate costs are all equal to some constant value, or are easily computed using some auxiliary model. However, there is no reason that ARTDP could not be extended to build a model of the immediate cost function as well.

Notice that the action selection procedure (line 7) is left unspecified. Unlike RTDP, ARTDP can not always choose what appears to be the best, or greedy action. This is because it only has a *model* of the state transition probabilities on which to base its decisions, and the model could be quite inaccurate initially. An inaccurate model could cause ARTDP to overestimate the values (long term costs) of some states. This in turn could cause actions that lead to those states to be ignored, even though they would be seen to be the optimal actions if P were known accurately. ARTDP needs to explore, to choose actions that do not currently appear to be optimal, in

```
1   Set $k = 0$.
2   Initialize counts and $\hat{P}$.
3   repeat forever {
4       Set $x_k$ to be a randomly selected start state.
5       while ($x_k$ is not a goal state) {
6           Perform the backup operation.

            $V_{k+1}(x) = \begin{cases} \min_{a \in \mathcal{A}(x)} \sum_{y \in \mathcal{X}} \hat{P}(x,y,a) \Big[ \; R(x,y,a) + \gamma V_k(y) \; \Big] & \text{if } x \in B_k \\ V_k(x) & \text{otherwise} \end{cases}$

7           Select an action, $a_k$.
8           Perform action and get next state, $x_{k+1}$.
9           $\eta(x_k, x_{k+1}, a_k) = \eta(x_k, x_{k+1}, a_k) + 1$
10          $\eta(x_k, x_{k+1}) = \eta(x_k, x_{k+1}) + 1$
11          Renormalize the transition probability estimates for all actions $a \in \mathcal{A}(x_k)$

                $\hat{P}(x_k, x_{k+1}, a) = \eta(x_k, x_{k+1}, a) / \eta(x_k, x_{k+1})$.

12          $k = k + 1$
13      }
14  }
```

Figure 3.5 Trial-based Adaptive Real-Time Dynamic Programming. Line 4 is the beginning of a trial. k counts the total number design/execution phases since the algorithm was started. The action selection procedure (line 7) should not always choose the *best* action, as RTDP does. Some exploration must be performed to ensure that $\hat{P} \rightarrow P$ over time.

order to ensure that $\hat{P} \rightarrow P$ over time. Gullapalli and Barto [32] prove that ARTDP will converge asymptotically to the optimal value function (and the optimal policy) with probability 1, provided that every admissible action is taken from every state an infinite number of times. Section 3.7 will discuss exploration in more depth.

Adaptive RTDP is related to a number of algorithms that have been investigated by others. Although Sutton's *Dyna* architecture [85] focuses on Q-learning and methods based on policy iteration (Section 2.2.2.3), it also encompasses algorithms such as ARTDP, as discussed in Sutton [87]. Lin [51,52] also discusses methods closely related to ARTDP. In the engineering literature, Jalali and Ferguson [39,40] describe an algorithm that is similar to ARTDP, although they focus on Markovian decision problems in which performance is measured by the average cost per-time-step instead of the discounted cost we have discussed. See also Schwartz [74, 75] for a further discussion of IDP algorithms for the average cost objective function.

3.6 Real-Time Q-learning

Q-learning does not use a model of the MDP to form its estimates of the optimal value function or the optimal policy. Therefore, a comparison of the performance of Q-learning with the performances of RTDP and ARTDP will provide some understanding of the advantages to be gained from using a model. Figure 3.6 describes Trial-based Real-Time Q-learning (RTQ) algorithm. Unlike the description of the basic Q-learning algorithm given in Figure 2.5, which allows arbitrary selection of the state x_k (line 2), this algorithm is like RTDP and ARTDP in following a sample trajectory through the state space. Also like ARTDP, RTQ must include some element of exploration in its action selection procedure (line 5). Following from the general convergence results for Q-learning (Section 2.2.4), RTQ will converge to the optimal Q-function and the optimal policy for all states when applied to undiscounted stochastic shortest path problems under the following conditions: 1) there exists a proper stationary policy; 2) the exploration procedure is properly designed so that all admissible actions are performed for all states infinitely often in an infinite sequence of trials; and 3) the learning rate is reduced appropriately. The first condition merely ensures that $V^*(x)$ is finite for all states, or equivalently, that there is a path from every state to the goal set.

RTDP and ARTDP can use their model of the MDP to take advantage of any looseness in the real-time constraints. If there is time, they can update the value estimates for some set of states B_k, of which the current state, x_k, is only one element. The states in B_k could be chosen by some method such as *prioritized sweeping* [61,62], which is designed to accelerate convergence of the value function. Since RTQ does not have a model of the MDP, it is unable to take advantage of any such scheme. RTQ can only update the value of a single (state, action) pair at each time step.

3.7 Exploration

Any exploration strategy for an on-line algorithm must try to balance two competing goals: exploration and exploitation. Efficient exploration requires that actions should always be chosen so that the average useful information gain per time step is maximized. However, since this is an on-line algorithm, it is also desirable to exploit what is currently known about the problem in order to find a solution. One popular exploration method that attempts to make this tradeoff is based on the Boltzmann distribution, and has been used, for example, by Watkins [92], Lin [52], and Sutton [85]. We use this exploration strategy in conjunction with ARTDP and RTQ in the experiments described in Section 3.8.

This method assigns an execution probability to each admissible action for the current state, where this probability is determined by a rating of each action's utility. We compute a rating, $r(a)$, of each action $a \in \mathcal{A}(x_k)$ as follows, depending on whether a model of the MDP is available. For Q-learning, which has no model, $r(a)$ is computed as

$$r(a) = Q_k(x_k, a), \tag{3.1}$$

```
1  Set $k = 0$.
2  **repeat** forever {
3      Set $x_k$ to be a randomly selected start state.
4      **while** ($x_k$ is not a goal state) {
5          Choose an action $a_k$ from $\mathcal{A}(x_k)$.
6          Execute action $a_k$ from state $x_k$. This results in a stochastic transition
           to state $x_{k+1}$, and incurs the cost $r_k = R(x_k, x_{k+1}, a_k)$.
7          Update the estimate $Q_k$ using the update rule given in Equation (2.22)

           $$Q_{k+1}(x_k, a_k) = Q_k(x_k, a_k) + \alpha \left[ r(x_k, x_{k+1}, a_k) + \gamma \min_{b \in \mathcal{A}(x_{k+1})} [Q_k(x_{k+1}, b)] - Q_k(x_k, a_k) \right]$$

8          $k = k + 1$
9      }
10 }
```

Figure 3.6 Trial-based Real-Time Q-learning. Line 3 is the beginning of a trial. k counts the total number design/execution phases since the algorithm was started. Unlike the description of the basic Q-learning algorithm given in Figure 2.5, which allows arbitrary selection of the state x_k (line 2), this algorithm follows a sample trajectory through the state space. The action selection procedure (line 5) should not always choose the *best* action, as RTDP does. Some exploration must be performed to ensure that the convergence assumption, that all admissible actions are performed for all states infinitely often, is satisfied.

where the function Q_k is the current estimate of the optimal Q-function, Q^*. For ARTDP, which builds a model of the MDP, $r(a)$ is computed as

$$r(a) = \sum_{y \in \mathcal{X}} \left[\hat{P}(x, y, a) R(x, y, a) \right] + \gamma \sum_{y \in \mathcal{X}} \left[\hat{P}(x, y, a) V_k(y) \right], \tag{3.2}$$

where V_k is the current estimate for the optimal value function, V^*. The latter definition for $r(a)$ is based on Equation (2.5), which defines the Q-function in terms of the value function. These ratings define a probability mass function over the admissible actions using the Boltzmann distribution: the probability that the controller executes action $a \in \mathcal{A}(x_k)$ is

$$\text{Prob}(a) = \frac{e^{-r(a)/T}}{\sum_{b \in \mathcal{A}(x_k)} e^{-r(b)/T}}, \tag{3.3}$$

where T is a positive parameter controlling how sharply these probabilities peak at the certainty equivalence optimal action. As T increases, these probabilities become more uniform, and as T decreases, the probability of executing the certainty equivalence optimal action approaches one, while the probabilities of the other actions approach zero. T acts as a kind of "computational temperature" as used in simulated annealing [45] in which T decreases over time. Here it controls the necessary tradeoff between exploration and exploitation. At "zero temperature" there is no exploration, and the randomized policy equals the certainty equivalence optimal policy, whereas at "infinite temperature" there is no attempt at exploitation, and actions are chosen from a uniform distribution.

3.8 Experiments

3.8.1 The Race Track Problem

We use a version of a game called Race Track, described by Martin Gardner [27], to test and compare the performance of conventional DP and the on-line algorithms RTDP, ARTDP, and RTQ. Race Track is a discrete-time, discrete-state simulation of automobile racing. We modify the game by allowing only a single player, and by making it probabilistic. It would be possible to allow multiple players, but that is unnecessary for demonstration purposes.

A race track of any shape is drawn on graph paper, with a starting line at one end and a finish line at the other consisting of designated squares. Each square within the boundary of the track is a possible location of the car. Figure 3.7 shows two example tracks. At the start of each trial, the car is placed at a random position on the starting line, and actions are selected in an attempt to move the car down the track toward the finish line. Acceleration and deceleration are simulated as follows. If in the previous move the car moved h squares horizontally and v squares vertically, then the present move can be h' squares vertically and v' squares horizontally, where the difference between h' and h is -1, 0, or 1, and the difference between v' and v is -1, 0, or 1. This means that the car can maintain its speed in either dimension, or it can slow down or speed up in either dimension by one square per move. If the car hits the track boundary,[3] we move it back to a random position on the starting line, reduce its velocity to zero (i.e., $h' - h$ and $v' - v$ are considered to be zero), and continue the trial. The objective is to learn to control the car so that it crosses the finish line in as few moves as possible. Figures 3.8 and 3.10 show examples of optimal and near-optimal paths for the race tracks shown in Figure 3.7.

We now introduce a random element into the problem. With a probability p, the actual accelerations or decelerations at a move are zero independently of the intended accelerations or decelerations. Thus, $1 - p$ is the probability that the controller's intended actions are executed. One might think of this as simulating driving on a track that is unpredictably slippery so that sometimes braking and throttling up have no effect on the car's velocity.

In order to describe the Race Track problem as a stochastic shortest path problem, we must formalize the description of the states, the actions, the transition probabilities, the immediate cost function, the start states, and the goal states. The state of the system at time k is the vector $(x_k, y_k, \dot{x}_k, \dot{y}_k)$. This is the position on the

[3]For the computational experiments described in Section 3.8.2, this means that the projected path of the car for a move intersects the track boundary at any place not on the finish line.

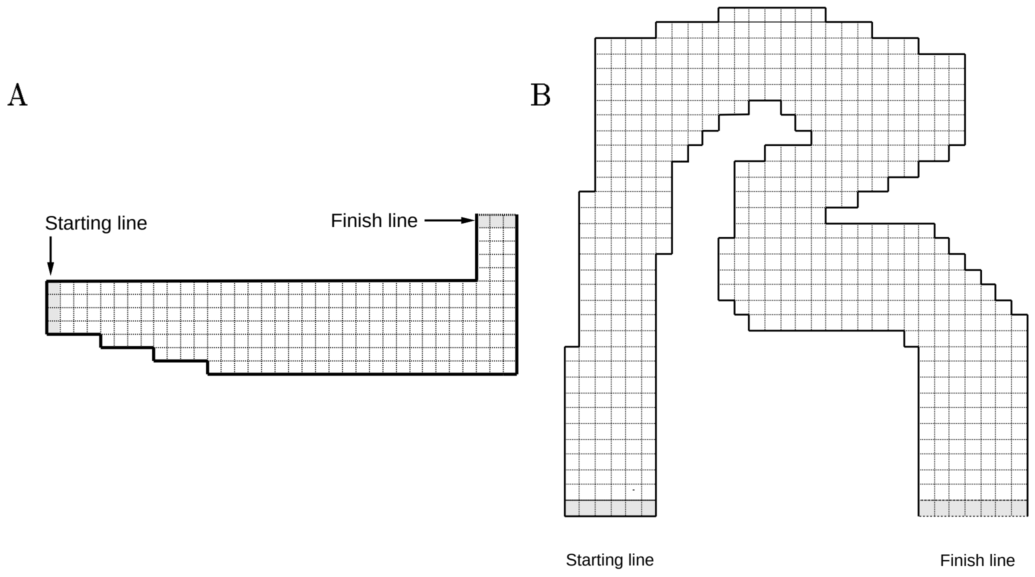


Figure 3.7 Example Race Tracks. Panel A: Small race track. Panel B: Larger race track. See Table 3.1 for details.

track at time k (x_k and y_k), and the velocity at time k ($\dot{x}_k$ and $\dot{y}_k$). The velocity at time k is the difference between the current position and the previous position, so $\dot{x}_k = x_k - x_{k-1}$, and $\dot{y}_k = y_k - y_{k-1}$. The velocity at the start of a trial is a special case, and is defined to be zero in both the x and y directions. The admissible actions are pairs (a^x, a^y), where $a^x \in \{-1, 0, 1\}$ is a velocity increment in the x direction, and where $a^y \in \{-1, 0, 1\}$ is a velocity increment in the y direction. All actions are admissible in all states.

The following equations define the state transitions of this system. With probability $1-p$, the controller's action is reliably executed so that the state at time step $k+1$ is

$$x_{k+1} = x_k + \dot{x}_k + a^x_k \tag{3.4a}$$

$$y_{k+1} = y_k + \dot{y}_k + a^y_k \tag{3.4b}$$

$$\dot{x}_{k+1} = \dot{x}_k + a^x_k \tag{3.4c}$$

$$\dot{y}_{k+1} = \dot{y}_k + a^y_k, \tag{3.4d}$$

and with probability p, the system ignores the controller's action, so that the state at time step $k+1$ is

$$x_{k+1} = x_k + \dot{x}_k \tag{3.5a}$$

$$y_{k+1} = y_k + \dot{y}_k \tag{3.5b}$$

$$\dot{x}_{k+1} = \dot{x}_k \tag{3.5c}$$

$$\dot{y}_{k+1} = \dot{y}_k. \tag{3.5d}$$

This assumes that the straight line joining the point (x_k, y_k) to the point (x_{k+1}, y_{k+1}) lies entirely within the track, or intersects only the finish line. If this is not the case, then the car has collided with the track's boundary, and the state at time $k+1$ is $(x, y, 0, 0)$, where (x, y) is a randomly chosen position on the starting line. A move that takes the car across the finish line is treated as a valid move, but we assume that the car subsequently stays in the resulting state until a new trial begins. This method for keeping the car on the track, together with Equations (3.4) and (3.5), define the state-transition probabilities for all states and admissible actions.

The total number of states for any racetrack problem is potentially infinite since we have not imposed a limit on the car's speed. However, in practice, the speed is limited by the size of the track. Therefore, the set of states that can be reached from the set of start states via any policy is finite and can be considered to be the state set of the stochastic shortest path problem.

To complete the formulation of the stochastic shortest path problem, we need to define the set of start states, the set of goal states, and the immediate costs associated with each action in each state. The set of start states consists of all the zero-velocity states on the starting line, i.e., all the states $(x, y, 0, 0)$ where (x, y) are coordinates of the squares making up the starting line. The set of goal states consists of all states that can be reached in one time step by crossing the finish line from inside the track. According to the state-transition function defined above, this set is absorbing. The immediate cost for all non-goal states is one independently of the action taken, i.e., $R(x, a) = 1$ for all non-goal states x and all admissible actions a. The immediate cost associated with a transition from any goal state is zero. With $R(x, a) = 1$, an optimal policy is one that minimizes both the expected time (number of steps) and the expected number of states visited from start to goal. The function R could, in general, be defined arbitrarily.

3.8.2 Results

We used the race track problem described in Section 3.8.1 to illustrate and compare conventional DP, RTDP, ARTDP, and RTQ using the two race tracks shown in Figure 3.7. The small race track shown in Panel A has 4 start states, 87 goal states, and $9,115$ states reachable from the start states by any policy. We have not shown the squares on which the car might land after crossing the finish line. The larger race track shown in Panel B has 6 start states, 590 goal states, and $22,576$ states reachable from the start states. We set $p = 0.1$ so that the controller's intended actions were executed with probability 0.9.

We applied conventional Gauss-Seidel DP to each race track problem, by which we mean Gauss-Seidel value iteration as defined in Section 2.2.2.2, with $\gamma = 1$ and with the initial evaluation function assigning zero cost to each state. Gauss-Seidel DP converges under these conditions because it is a special case of asnychronous DP, which converges here because the conditions given in Section 2.2.3 are satisfied. Specifically, it is clear that there is at least one proper policy for either track (it is possible for the car to reach the finish line from any reachable state, although it may have to hit the wall and restart to do so) and every improper policy incurs infinite cost for at least one state because the immediate costs of all non-goal states are positive. We selected a state ordering for applying Gauss-Seidel DP without concern for any influence it might have on convergence rate (although we found that with the selected ordering, Gauss-Seidel DP converged in approximately half the number of sweeps as did Jacobi DP).

Table 3.1 summarizes the small and larger race track problems and the computational effort required to solve them using Gauss-Seidel DP. Gauss-Seidel DP was considered to have converged to the optimal evaluation function when the maximum cost change over all states between two successive sweeps was less than 10^{-4}. We estimated the number of relevant states for each race track, i.e., the number of states reachable from the start states under any optimal policy, by counting the states visited while executing optimal actions for 10^7 trials.

We also estimated the earliest point in the DP computation at which the optimal evaluation function approximation was good enough so that the corresponding greedy policy was an optimal policy. (Recall from Section 2.2.1 that an optimal policy can be a greedy policy with respect to many evaluation functions.) We did this by running 10^7 test trials after each sweep using a policy that was greedy with respect to the evaluation function produced by that sweep. For each sweep, we recorded the average path length produced over these test trials. After convergence of Gauss-Seidel DP, we compared these averages with the optimal expected path length obtained by the DP algorithm, noting the sweep after which the average path length was first within 10^{-2} of the optimal. The resulting numbers of sweeps and backups are listed in Table 3.1 in the columns labeled "Number of GSDP sweeps to optimal policy" and "Number of GSDP backups to optimal policy." Although optimal policies emerged considerably earlier in these computations than did the optimal evaluation functions, it is important to note that this estimation process is not a part of conventional off-line value iteration algorithms and requires a considerable amount of additional computation.[4] Nevertheless, the resulting numbers of backups are useful in assessing the computational requirements of the real-time algorithms, which should allow controllers to follow optimal policies after comparable numbers of backups.

We applied RTDP, ARTDP, and RTQ to both race track problems. Because all the immediate costs are positive, we know that $V^*(x)$ must be non-negative for all states x. Thus, setting the initial costs of all the states to zero produces a non-overestimating initial evaluation function as required by Theorem 3.3. We applied the real-time algorithms in a trial-based manner, starting each trial with the car placed on the starting line with zero velocity, where each square on the starting line was selected with equal probability. A trial ended when the car reached a goal state. Thus, according to Theorem 3.3, with $\gamma = 1$, RTDP will converge to the optimal evaluation function with repeated trials. Although RTDP and ARTDP can back up the costs of many states at each control step, we restricted attention to the simplest

[4]Policy iteration algorithms address this problem by explicitly generating a sequence of improving policies, but updating a policy requires computing its corresponding evaluation function, which is generally a time-consuming computation.

Table 3.1 Example Race Track Problems. The results were obtained by executing Gauss-Seidel DP (GSDP).

	Small track	*Larger track*
Number of reachable states	9,115	22,576
Number of goal states	87	590
Est. number of relevant states	599	2,618
Optimum exp. path length	14.67	24.10
Number of GSDP sweeps to convergence	28	38
Number of GSDP backups to convergence	252,784	835,468
Number of GSDP sweeps to optimal policy	15	24
Number of GSDP backups to optimal policy	136,725	541,824

case in which they only back up the cost of the current state at each time step. This is the case in which $B_k = \{s_k\}$ for all k.

We executed 25 runs of each algorithm using different random number seeds, where a run is a sequence of trials beginning with the evaluation function initialized to zero. To monitor the performance of each algorithm, we kept track of path lengths, that is, how many moves the car took in going from the starting line to the finish line, in each trial of each run. To record these data, we divided each run into a sequence of disjoint *epochs*, where an epoch is a sequence of 20 consecutive trials. By an *epoch path length* we mean the average of the path lengths generated during an epoch using a given algorithm. ARTDP and RTQ were applied under conditions of incomplete information, and for these algorithms we induced exploratory behavior by using randomized policies based on the Boltzmann distribution as described in Section 3.7. To control the tradeoff between identification and control, we decreased the parameter T in Equation (3.3) after each move until it reached a pre-selected minimum value; T was initialized at the beginning of each run. Parameter values and additional simulation details are provided in Appendix B.

Figure 3.8 shows results for RTDP (Panel A), ARTDP (Panel B), and RTQ (Panel C). The central line in each graph shows the epoch path length averaged over the 25 runs of the corresponding algorithm. The upper and lower lines show ± 1 standard deviation about this average for the sample of 25 runs. Although the average epoch path lengths for the initial several epochs of each algorithm are too large to show on the graphs, it is useful to note that the average epoch path lengths for the first epoch of RTDP, ARTDP, and RTQ are respectively 455, 866, and 13,403 moves. That these initial average path lengths are so large, especially for RTQ, reflects the primitive nature of our exploration strategy.

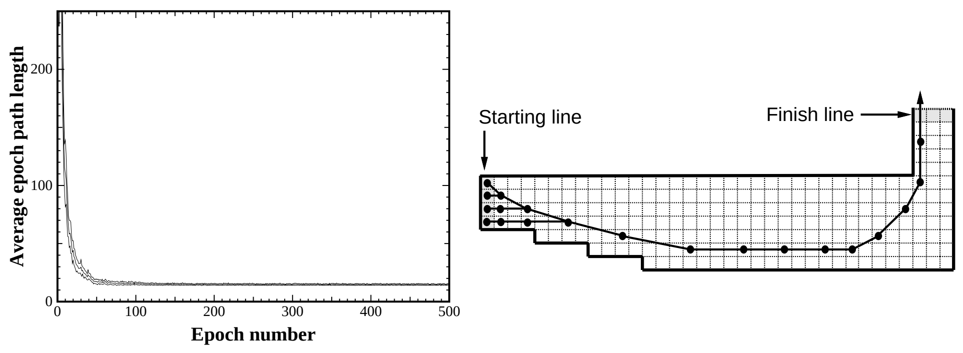
It is clear from the graphs that in this problem RTDP learned faster (and with less variance) than ARTDP and RTQ, when learning rate is measured in terms of the number of epochs (numbers of moves are given in Table 3.2 discussed below). This is not surprising given the differences between the versions of the problem with complete information (Panel A) and with incomplete information (Panels B and C). That the performances of RTDP and ARTDP were so similar despite these differences reflects the fact that the maximum likelihood system identification procedure used by the latter algorithm converged rapidly on relevant states due to the low level of stochasticity in the problem ($p = 0.1$). These graphs also show that RTQ takes very many more epochs than do RTDP and ARTDP to reach a similar level of performance. This reflects the fact that each backup in RTQ takes into account less information than do the backups in RTDP or ARTDP, a disadvantage somewhat offset by the relative computational simplicity of each Q-learning backup. Figure 3.9 shows the RTQ results out to 5,000 epochs.

A convenient way to show the policies that result from these algorithms is to show the paths the car would follow from each start state if all sources of randomness were turned off; that is, if both random exploration and the randomness in the problem's state transition function were turned off. At the right in each panel of Figure 3.8 are paths generated in this way by the policies produced after each algorithm was judged to have "effectively converged." We inspected the graphs to find the smallest epoch numbers at which the average epoch path lengths essentially reached their asymptotic levels: 200 epochs for RTDP (Panel A), 300 epochs for ARTDP (Panel B), and 2,000 epochs for RTQ (Panel C). Treated with appropriate caution, these effective convergence times are useful in comparing algorithms.

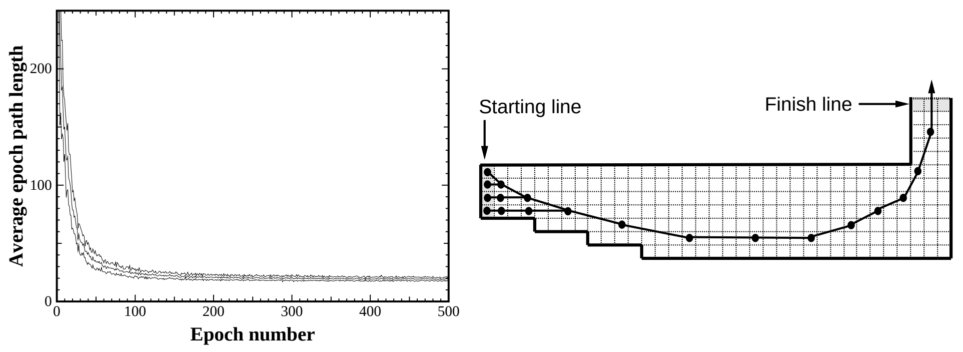
The path shown in Panel A of Figure 3.8 is optimal in the sense that it was produced in noiseless conditions by a policy that is optimal for the stochastic problem. The paths in Panels B and C, on the other hand, were not generated by an optimal policy despite the fact that each is one move shorter than the path of Panel A. The control decisions made toward the end of the track by these suboptimal policies produce higher probability that the car will collide with the track boundary under stochastic conditions. Although we do not illustrate it here, as the amount of uncertainty in the problem increases (increasing p), optimal policies generate paths that are more "conservative" in the sense of keeping safer distances from the track boundary and maintaining lower velocities.

Table 3.2 provides additional information about the performance of the real-time algorithms on the small track. For comparative purposes, the table includes a column for Gauss-Seidel DP. We estimated the path length after the effective convergence of RTDP, ARTDP, and RTQ by executing 500 test trials with learning turned off using the policy produced at effective convergence of each algorithm. We also turned off the random exploration used by the latter two algorithms. The row of Table 3.2 labeled "Est. path length at effective convergence" gives the average path length

A



B



C

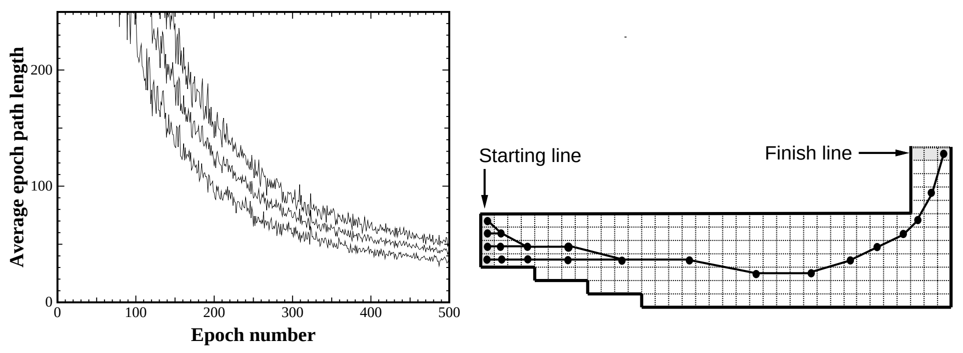


Figure 3.8 Performance of Three Real-Time Learning Algorithms on the Small Track. Panel A: RTDP. Panel B: ARTDP. Panel C: RTQ. The central line in each graph shows the epoch path length averaged over the 25 runs of the corresponding algorithm. The upper and lower lines show ± 1 standard deviation of the epoch path length for the sample of 25 runs. Exploration was controlled for ARTDP and RTQ by decreasing T after each move until it reached a pre-selected minimum value. The right side of each panel shows the paths the car would follow in noiseless conditions from each start state after effective convergence of the corresponding algorithm.

Table 3.2 Summary of Learning Performance on the Small Track for RTDP, ARTDP, and RTQ. The amount of computation required by Gauss-Seidel DP (GSDP) is included for comparative purposes.

	GSDP	*RTDP*	*ARTDP*	*RTQ*
Ave. time to effective convergence	28 sweeps	200 epochs	300 epochs	2, 000 epochs
Est. path length at effective convergence	14.56	14.83	15.10	15.44
Ave. number of backups	252, 784	127, 538	218, 554	2, 961, 790
Ave. number of backups per epoch	-	638	728	1, 481
% of states backed up ≤ 100 times	-	98.45	96.47	53.34
% of states backed up ≤ 10 times	-	80.51	65.41	6.68
% of states backed up 0 times	-	3.18	1.74	1.56

over these test trials.[5] RTDP is most directly comparable to Gauss-Seidel DP. After about 200 epochs, or 4,000 trials, RTDP improved control performance to the point where a trial took an average of 14.83 moves. RTDP performed an average of 127,538 backups in reaching this level of performance, about half the number required by Gauss-Seidel DP to converge to the optimal evaluation function. This number of backups is comparable to the 136,725 backups in the 15 sweeps of Gauss-Seidel DP after which the resulting evaluation function defines an optimal policy (Table 3.1).

Another way to compare Gauss-Seidel DP and RTDP is to examine how the backups they perform are distributed over the states. Whereas the cost of every state was backed up in each sweep of Gauss-Seidel DP, RTDP focused backups on fewer states. For example, in the first 200 epochs of an average run, RTDP backed up the costs of 98.45% of the states no more than 100 times and 80.51% of the states no more than 10 times; the costs of about 290 states were not backed up at all in an average run. Although we did not collect these statistics for RTDP after 200 epochs, it became even more focused on the states on optimal paths.

Not surprisingly, solving the problem under conditions of incomplete information requires more backups. ARTDP took 300 epochs, or an average of 218,554 backups, to achieve trials averaging 15.1 moves at effective convergence. RTQ took 2,000

[5]These path length estimates are somewhat smaller than the average epoch path lengths shown at effective convergence in the graphs of Figure 3.8 because they were produced with exploration turned off, whereas the graphs show path lengths produced with random exploration turned on. For Gauss-Seidel DP, we averaged over the costs of the start states given by the computed optimal evaluation function to obtain the estimated path length listed in Table 3.2.

epochs, or an average of 2,961,790 backups, to achieve a somewhat less skillful level of performance (see Figure 3.9). Examining how these backups were distributed over states shows that ARTDP was considerably more focused than was RTQ. In the first 300 epochs ARTDP backed up 96.47% of the states no more than 100 times and 65.41% of the states no more than 10 times. On the other hand, in 2,000 epochs RTQ backed up Q-values for 53.34% of the states no more than 100 times and only 6.68% of the states no more than 10 times.[6] Again, these results for RTQ reflect the inadequacy of our primitive exploration strategy for this algorithm.

Figure 3.10 shows results for RTDP, ARTDP, and RTQ on the larger race track, and Table 3.3 provides additional information. These results were obtained under the same conditions described above for the small track. Figure 3.11 shows the RTQ results for the larger track out to 7,500 epochs. We judged that RTDP, ARTDP, and RTQ had effectively converged at 500, 400, and 3,000 epochs respectively. That ARTDP effectively converged faster than RTDP in terms of the number of epochs is partially due to the fact that its epochs tended to have more moves, and hence more backups, than the epochs of RTDP. We can see that to achieve slightly suboptimal performance, RTDP required about 62% of the computation of conventional Gauss-Seidel DP. The average epoch path lengths for the initial epoch of each algorithm, which are too large to show on the graphs, are 7,198, 8,749, and 180,358 moves, respectively, for RTDP, ARTDP, and RTQ. Again, these large numbers of moves, especially for RTQ, reflect the primitive nature of our exploration strategy. The paths shown at the right in each panel of Figure 3.10 were generated in noiseless conditions by the policies produced at effective convergence of the corresponding algorithms. The path shown in Panel A of Figure 3.10 is optimal in the sense that it was produced in noiseless conditions by a policy that is optimal for the stochastic problem. The paths in Panels B and C, on the other hand, were generated by slightly suboptimal policies.

Although these simulations are not definitive comparisons of the real-time algorithms with conventional DP, they illustrate some of their features. Whereas Gauss-Seidel DP continued to back up the costs of all the states, the real-time algorithms strongly focused on subsets of the states that were relevant to the control objectives. This focus became increasingly narrow as learning continued. Because the convergence theorem for Trial-Based RTDP applies to the simulations of RTDP, we know that this algorithm eventually would have focused only on relevant states, i.e., on states making up optimal paths. RTDP achieved nearly optimal control performance with about 50% of the computation of Gauss-Seidel DP on the small track and about 62% of the computation of Gauss-Seidel DP on the larger track. ARTDP and RTQ also focused on progressively fewer states, but less efficiently, as the exploration strategy were required continued selection of actions that RTDP learned to ignore.

The results described here for ARTDP and RTQ were produced by using an exploration strategy that decreased the randomness in selecting actions by decreasing

[6]We considerated a Q-value for a state x to be backed up whenever $Q(x, a)$ was updated for some $a \in \mathcal{A}(x)$.

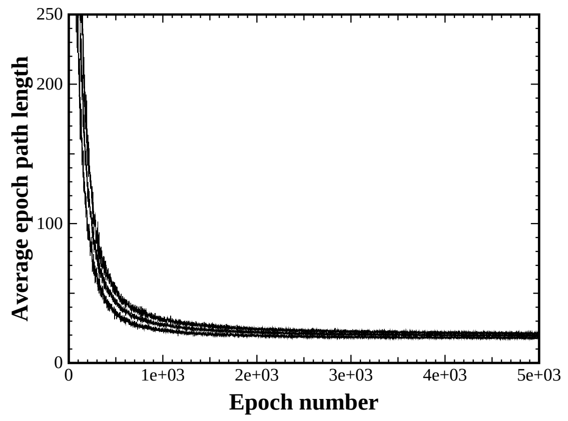
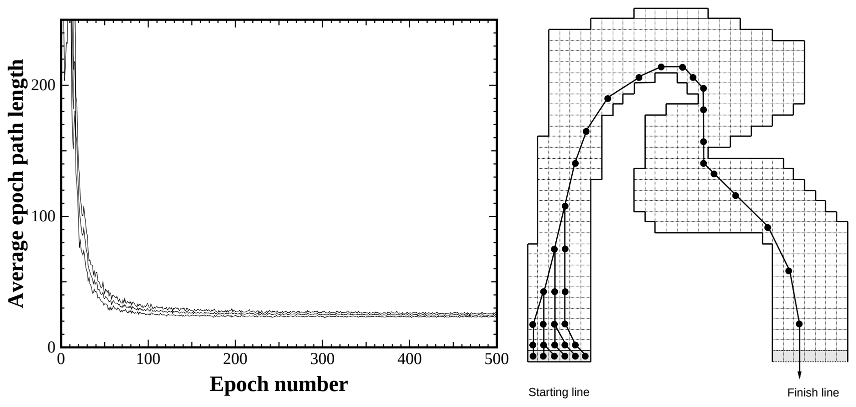


Figure 3.9 Performance of RTQ on The Small Track for 5,000 Epochs. The initial part of the graph shows the data plotted in Panel C of Figure 3.8 but at a different horizontal scale.

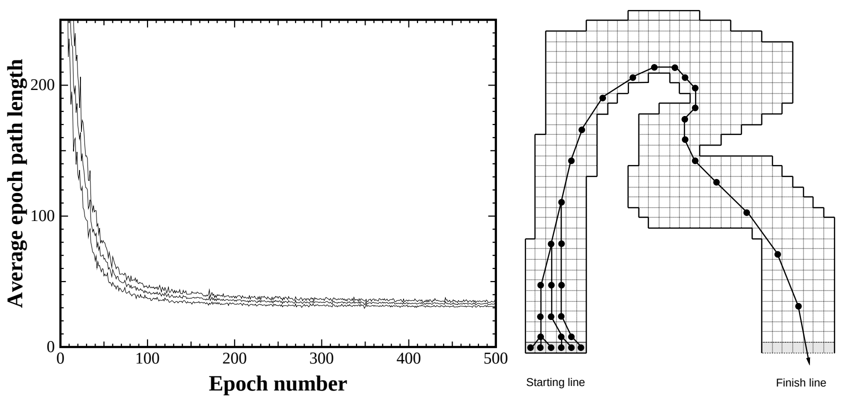
Table 3.3 Summary of Learning Performance on the Larger Track for RTDP, ARTDP, and RTQ. The amount of computation required by Gauss-Seidel DP (GSDP) is included for comparative purposes.

	GSDP	*RTDP*	*ARTDP*	*RTQ*
Ave. time to effective convergence	38 sweeps	500 epochs	400 epochs	$3,000$ epochs
Est. path length at effective convergence	24.10	24.62	24.72	25.04
Ave. number of backups	$835,468$	$517,356$	$653,774$	$10,330,994$
Ave. number of backups per epoch	-	$1,035$	$1,634$	$3,444$
% of states backed up ≤ 100 times	-	97.77	90.03	52.43
% of states backed up ≤ 10 times	-	70.46	59.90	8.28
% of states backed up 0 times	-	8.17	3.53	2.70

A



B



C

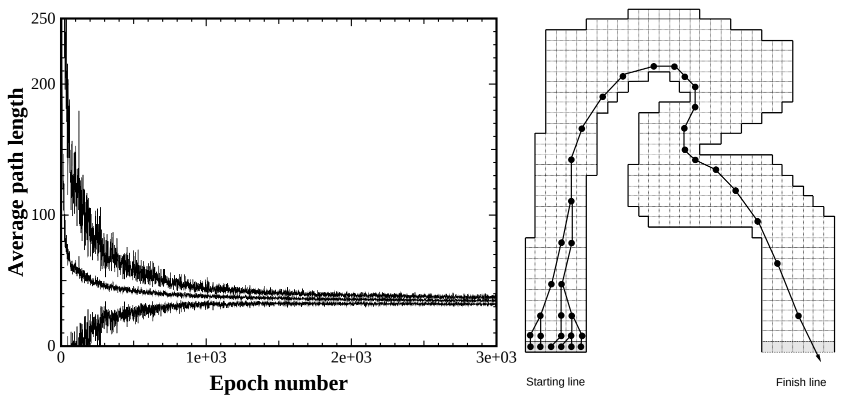


Figure 3.10 Performance of Three Real-Time Learning Algorithms on the Larger Track. Panel A: RTDP. Panel B: ARTDP. Panel C: RTQ. The central line in each graph shows the epoch path length averaged over the 25 runs of the corresponding algorithm. The upper and lower lines show ± 1 standard deviation of the epoch path length for the sample of 25 runs. Exploration was controlled for ARTDP and RTQ by decreasing T after each move until it reached a pre-selected minimum value. The right side of each panel shows the paths the car would follow in noiseless conditions from each start state after effective convergence of the corresponding algorithm.

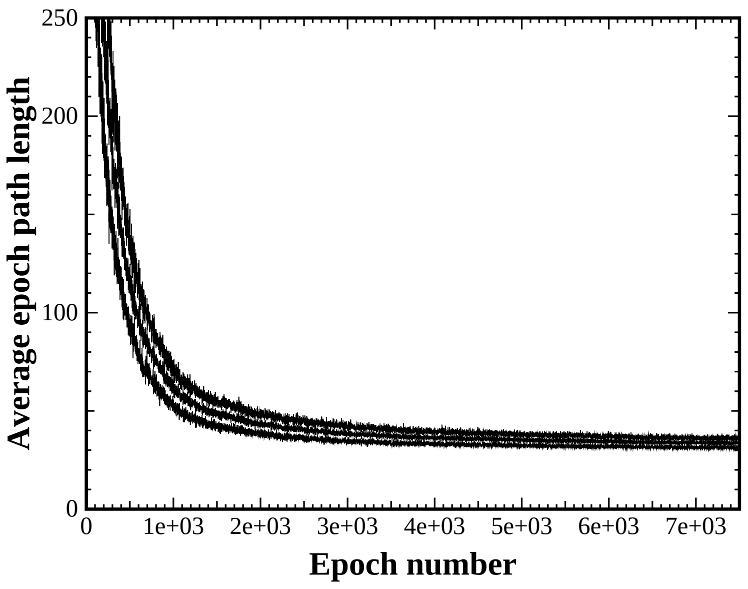


Figure 3.11 Performance of RTQ on The Larger Track for 7,500 Epochs. The initial part of the graph shows the same data as plotted in Panel C of Figure 3.10 but at a different horizontal scale.

T after each move until it reached a pre-selected minimum value. Although not described here, we also conducted experiments with different minimum values and with decreasing T after trials instead of after moves. Performance of the algorithms was much altered (for the worse) by these changes. Although we made no systematic attempt to investigate the effects of various exploration strategies, it is clear that the performance of these algorithms is highly sensitive to how exploration is introduced and controlled.

3.9 Iterative Deepening Real-time Dynamic Programming

The algorithms presented above (trial-based RTDP, ARTDP, and RTQ), when applied to an undiscounted stochastic shortest path problem, are guaranteed to converge to the optimal value function and optimal policy on the relevant states provided that there is as least one proper policy. If there is no proper policy, then it would be possible for these algorithms to enter a trap state (or ergodic set of trap states), from which there is no path to the goal set. But there may be a *partial proper* policy with respect to the start states. A policy U is a partial proper policy with respect to some set of states $\mathcal{S} \in \mathcal{X}$ if following U from any state in $\mathcal{S}$ leads eventually, with probability one, to the goal set[7]. Iterative-deepening RTDP (IDRTDP) is a modification of trial-based RTDP which weakens the requirement for the existance of a proper policy. IDRTDP only requires the existance of a partial proper policy with respect to the start states. Figure 3.12 gives a pseudocode description of IDRTDP. ARTDP and RTQ could be modified similarly. The term "iterative-deepening" is

[7]A policy U is a proper policy iff it is a partial proper policy with respect to every subset of $\mathcal{X}$.

used through analogy to Korf's [47] Iterative-Deepening A*, which uses a similarly increasing depth bound.

IDRTDP uses the length of the current trial as a test to see if it might have fallen into a trap set. If the length of the current trial exceeds the maximum allowed trial length, then the current trial is terminated and the maximum allowed trial length is incremented. A new trial begins from a randomly chosen start state. The length bound is necessary. Without it, the algorithm could get stuck in one of the trap sets before it learned to avoid them. It is important that the maximum allowed trial length progressively increases when IDRTDP is applied to stochastic shortest path problems. For one thing, the expected length of a trial cannot be known until the MDP has been solved. The expected length of a trial can be much greater than the total number of states when there are recurrent links, as demonstrated in Figure 3.13. There are only two states, the start state and the goal state, but the expected length of a trial is 100. The disparity between the number of states and the expected length of a trial can be made arbitrarily large by proper selection of the transition probabilities. If the maximum allowed length of a trial is fixed, and too short, then the goal set may be visited very rarely, if at all. A related reason for increasing the maximum allowed trial length is that it is possible to start from the start set and follow any partial proper policy for an unlimited number of steps before reaching the goal set. Trials of great length become increasingly unlikely as the length increases, but they are not ruled out. If we allow an infinite number of trials, we are guaranteed that there will be trials that will take more than any fixed number of steps. We do not want to stop a trial prematurely if we have not entered a trap set. The situation is simpler for a deterministic shortest path problem. In this case we know that the maximum distance from the start set to the goal set can be no more than $N = |\mathcal{X}|$ when following a partial proper policy.

Theorem 3.4 describes the conditions required for the convergence of IDRTDP. They are nearly the same as those given in Theorem 3.3 for RTDP. The only difference is that IDRTDP requires only a partial proper policy with respect to the start states, whereas RTDP requires a proper policy.

Theorem 3.4 *In undiscounted stochastic shortest path problems, Iterative Deepening RTDP, with the initial state of each trial restricted to a set of start states, converges (with probability one) to V^* on the set of relevant states, and the controller's policy converges to an optimal policy (possibly nonstationary) on the set of relevant states, under the following conditions: 1) there is at least one partial proper stationary policy with respect to the start states; 2) all immediate costs incurred by transitions from non-goal states are positive, i.e., $R(x, a) > 0$ for all non-goal states x and actions $a \in \mathcal{A}(x)$; and 3) the initial value estimates of all states are non-overestimating, i.e., $V_0(x) \leq V^*(x)$ for all states $x \in \mathcal{X}$.*

The types of problems that IDRTDP is designed to solve can be solved using the algorithms we discussed previously. However, some preprocessing (which IDRTDP does not require) is necessary in order to trim the problem to fit the requirements of those algorithms. Figure 3.14 describes one form of preprocessing that will adapt stochastic shortest path problems to the requirements of the algorithms discussed

```
Set $k = 0$.
Initialize $l$, the maximum allowed length of a trial.
**repeat** forever {
    Set $x_k$ to be a randomly selected start state.
    Set TrialDone = false.
    Set $t = 0$.
    **while** (TrialDone is false) {
        Perform the backup operation.

        $V_{k+1}(x) = \begin{cases} \min_{a \in \mathcal{A}(x)} \sum_{y \in \mathcal{X}} P(x,y,a) \Big[ \; R(x,y,a) + \gamma V_k(y) \; \Big] & \text{if } x \in B_k \\ V_k(x) & \text{otherwise} \end{cases}$

        Select the best action, given the *current estimate* of the optimal value
        function.

        $a_k = \underset{a \in \mathcal{A}(x_k)}{\text{argmin}} \sum_{y \in \mathcal{X}} P(x_k,y,a) \Big[ \; R(x_k,y,a) + \gamma V_{k+1}(y) \; \Big]$

        Perform action and get next state, $x_{k+1}$.
        **if** ($x_{k+1}$ is a goal state) {
            TrialDone = true.
        }
        **else if** ($t = l$) {
            TrialDone = true.
            Increment $l$.
        }
        Increment $k$ and $t$.
    }
}
```

Figure 3.12 Iterative Deepening RTDP. The Trial-based Iterative Deepening Real-Time Dynamic Programming algorithm. k counts the total number of design/execution phases since the algorithm was started. t counts the number of steps in the current trial. A trial is concluded when either a goal state is reached or when the trial has taken too long. If a trial times out, then the allowed maximum length of a trial is increased.

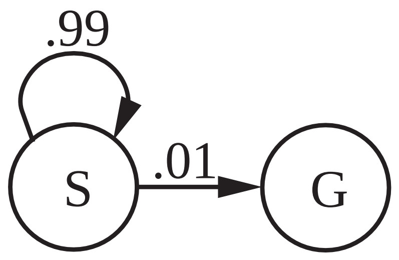


Figure 3.13 A recurrent stochastic shortest path problem. There is only one action and two states, the start state and the goal state. The probability of transition from S back to S is 0.99. The probability of transition from S to G is 0.01. The expected length of a trial from S to G is 100 steps.

previously. First, we determine the state connectivity. State x is connected to state y if there is a sequence of actions that leads from x to y with some probability greater than zero. State connectivity can be computed very efficiently. Next, we remove all states that are not connected to the goal set. This removes all trap states. Finally, we remove from consideration all actions that had a non-zero probability of leading to a trap state in one step. The result of these three steps is a stochastic shortest path problem with at least one proper policy (assuming that the original problem had at least one partial proper policy with respect to the start states). We can solve this truncated problem with the algorithm of choice. IDRTDP will be superior to this "truncate then solve" method when the interface between the non-trap and trap states is small enough. When this is true, IDRTDP will quickly learn never to make choices that lead into the trap states, and will focus its attention on the remaining states.

```
1  Determine the state connectivity.
2  Remove all states not connected to the goal set.
3  **repeat** until no further deletions {
4      For each remaining state $x$, for each action $a \in \mathcal{A}(x)$, remove $a$ from $\mathcal{A}(x)$ if
           $P(x, y, a) \neq 0$ for some state $y$ that has been removed.
5  }
6  Solve the remaining stochastic shortest problem using the algorithm of choice.
```

Figure 3.14 Off-line solution to partial proper policy problems.

IDRTDP is not suitable for use in the on-line solution of a problem where the trap states of the MDP correspond to catastrophic failure modes of some underlying real-world process. IDRTDP can not guarantee that the trap states will never be visited. See Musliner, Durfee, and Shin [63] or Schoppers [71] for other approaches to robust real-time control, where such guarantees are required of the system. IDRTDP is best used in simulation mode, finding a solution through interaction with the system model instead of through interaction directly with the real system.

3.10 Conclusions

In this chapter we developed several incremental DP algorithms (RTDP, ARTDP, and IDRTDP) that are suitable for real-time application. The standard, synchronous DP algorithms are too expensive to use as part of a real-time control scheme. Many researchers have studied this approach from a theoretical viewpoint (see, for example, Sato, Abe, and Takeda [70]). In test, however, these algorithms are too expensive for all but the smallest of problems. For example, the cost of one sweep through the state space for either Jacobi or Gauss-Seidel value iteration costs $O(mn)$ for a deterministic problem, and $O(mn^2)$ for a stochastic problem, where $n = |\mathcal{X}|$ and $m = |\mathcal{A}|$. This is a worst case analysis, because each state may have many fewer than n possible successors, and many fewer than m admissible actions. For the large state sets typical in AI and in many control problems, it is not desirable to try to complete even one iteration, let alone repeat the process until it converges to V^*. For example, because backgammon has about 10^{20} states, a single value iteration sweep through the state space would take more than 1,000 years using a 1,000 MIPS processor.

In general, none of RTDP, ARTDP, and IDRTDP are real time in the sense that they are guaranteed to complete their computations within a predetermined time limit. The reason for this is that in the general description of these algorithms, the definition of the set B_k is left unspecified except to say that the current state must be an element of B_k. However, B_k could be determined by a lookahead search such as employed by game-playing programs in an attempt to compute more accurate values for states along the probable state-space trajectory. The size of B_k could then be unknown before the search, and meaningful time bounds would be difficult to guarantee[8]. RTDP, ARTDP, and IDRTDP are better described as "any-time" algorithms as discussed by Dean and Boddy [25]. They each require some minimum amount of computation in order to update the value of the current state. After that, the process of updating the value function (and hence the policy) can be interrupted at any time and an answer, the value function estimate, will be available. There will be no disasterous consequences if the minimum time is not available during some update cycles. The value function will remain as it was, and will just need to be updated at some later time.

One of the difficulties involved in applying the algorithms discussed in this chapter to very large problems is caused by the amount of memory required to store the system model and the value or Q-function. For many problems of interest, there is no practical way to store the system model in an explicit, declarative form. Furthermore, the theory underlying these algorithms requires the value function, or the Q-function in the case of RTQ, to be represented as a lookup table. This representation rapidly

[8]Since there are only a finite number of states and actions, one could set a computation bound by the number of operations it would require for a complete sweep of the state set. Then, with a sufficiently fast processor, one could guarantee any time limit one wished. This is not a realistic method of guaranteeing real-time performance except for very small problems.

becomes impracticle. For an extreme example, again consider Backgammon, with its approximately 10^{20} states. It would be impossible to store just the value function as a lookup table. Explicit storage of the state transition probilities would require even greater resources. Tesauro's TD-Gammon system [89, 90] shows how these problems can be avoided. TD-Gammon avoids the problem of storing a very large model by representing the transition probabilities procedurally. The transition probabilities for each state and action are computed as needed, are used immediately, and are then discarded. This is an obvious solution, but it can only be applied to well understood problems where there are significant regularities to the pattern of transition probabilities. It is difficult to see how this approach could be used as part of the application of ARTDP to a large, unknown system. TD-Gammon avoids the problem of storing a lookup table representing the value function for 10^{20} states by using a neural network as a compact function approximation method. This reduces the space requirements by many orders of magnitude, but it is also outside the theory of incremental DP algorithms as it currently stands. Continued theoretical research is necessary to build an understanding of the conditions under which compact function approximation methods such as neural networks may be used as part of an incremental DP algorithm, and what impact their use will have on performance. Chapters 4 and 5 will further discuss these issues in the context of discrete MDPs and LInear-Quadratic Regulation, respectively.

CHAPTER 4

NEW ALGORITHMS FOR TEMPORAL DIFFERENCE LEARNING

In chapters 2 and 3 we assumed that we could use a lookup table to store the value function or Q-function that we wish to approximate. In this chapter we consider the use of linear function approximators in conjunction with several Temporal Difference (TD) learning rules. We start by reviewing the TD(λ) learning rule and the previously developed theory concerning its convergence. We describe three new TD algorithms, and show probability one convergence for their parameter estimates (under appropriate conditions). The first new algorithm, described in Section 4.4 is a normalized version of TD(λ). The second, described in Section 4.5, is developed using Least Squares and Instrumental Variable techniques. The third, also described in Section 4.5, is a recursive version of the second.

4.1 The TD(λ) learning rule

The TD(λ) learning rule [82] was developed as one of the results of a study of reinforcement learning algorithms and the problem of temporal credit assignment [1, 5, 81]. TD(λ) incrementally builds an estimate of the value function V_{U} for some given MDP and a fixed policy U. The selection of a policy reduces an MDP to a Markov chain since there is no longer any choice of the action to take at each state. Therefore, we will designate the value function estimated by TD(λ) as V instead of as V_{U}. Although TD(λ) can be used with lookup-table function representations, it is most often described in terms of parameterized approximators. Using our notation, summarized in Table 4.1, the TD(λ) learning rule for a differentiable parameterized function approximator (first described by Sutton [82]) is expressed as

$$\theta_{k+1} = \theta_k + \alpha_{\eta(x_k)} \Big[\ R_k + \gamma \hat{V}_k(x_{k+1}) - \hat{V}_k(x_k)\ \Big] \sum_{i=1}^{k} \lambda^{k-i} \nabla_{\theta_k} \hat{V}_k(x_i) \tag{4.1a}$$

$$= \theta_k + \alpha_{\eta(x_k)} \Delta\theta_k, \tag{4.1b}$$

where

$$\Delta\theta_k = \Big[\ R_k + \gamma \hat{V}_k(x_{k+1}) - \hat{V}_k(x_k)\ \Big] \sum_{i=1}^{k} \lambda^{k-i} \nabla_{\theta_k} \hat{V}_k(x_i) \tag{4.1c}$$

$$= \left[\; R_k + \gamma \hat{V}_k(x_{k+1}) - \hat{V}_k(x_k) \; \right] \Sigma_k \tag{4.1d}$$

and

$$\Sigma_k = \sum_{i=1}^{k} \lambda^{k-i} \nabla_{\theta_k} \hat{V}_k(x_i). \tag{4.1e}$$

Notice that $\Delta\theta_k$ depends only on estimates, $\hat{V}_k(x_i)$, made using the latest parameter values, θ_k. This is an attempt to separate the effects of changing the parameters from the effects of moving through the state space. However, when $\hat{V}$ is not a linear function of θ and $\lambda \neq 0$, the sum Σ_k cannot be formed in an efficient, recursive manner. Instead, it is necessary to remember x_i and to compute $\nabla_{\theta_k} \hat{V}_k(x_i)$ for all $i \leq k$. This is because, if $\hat{V}$ is nonlinear in θ, then $\nabla_{\theta_k} \hat{V}_k(x_i)$ will depend on θ_k. This implies that Σ_k cannot be defined recursively in terms of Σ_{k-1}. Recomputing Σ_k in this manner at every time step could be extremely expensive, so an approximation is usually used. Assuming that the learning rates are small, the difference between θ_k and θ_{k-1} will be small. Then an approximation to Σ_k can be defined recursively as

$$\Sigma_k = \lambda \Sigma_{k-1} + \nabla_{\theta_k} \hat{V}_k(x_k). \tag{4.1f}$$

If $\hat{V}$ *is* linear in θ, then Equation (4.1f) can be used to compute Σ_k exactly. No assumptions about the learning rates are required, and no approximations are made.

We will be concerned in the remainder of this chapter with linear function approximators, where $\hat{V}_k(x_i) = \phi_i' \theta_k$. In this case Equation (4.1f) becomes

$$\Sigma_k = \lambda \Sigma_{k-1} + \phi_k. \tag{4.1g}$$

4.2 Previous convergence results for TD(λ)

Convergence of the TD(λ) learning rule depends on the state representations and on the form of the function approximator. Although TD(λ) has been used successfully with nonlinear function approximators, most notably in Tesauro's self-teaching backgammon programs [89, 90], convergence has only been proven for lookup-table and linear function approximators.

The first proofs concerning the convergence of a version of TD(λ) were developed by Sutton [82] and Dayan [22], who showed parameter convergence in the mean, and by Dayan and Sejnowski [23], who showed parameter convergence with probability one. The learning algorithm they consider applies only to absorbing Markov chains. It is *trial-based*, with parameter updates only at the end of every trial. A trial is a sequence of states that starts at some start state, follows the Markov chain as it makes transitions, and ends at an absorbing state. The start state for each trial is chosen according to the probability distribution given by S. Figure 4.1 describes the algorithm. Since parameter updates take place only at the end of a trial, $\Delta\theta_k$ is defined somewhat differently from above:

$$\Delta\theta_k = \left[\; R_k + \gamma \theta_n' \phi_{k+1} - \theta_n' \phi_k \; \right] \sum_{i=1}^{k} \lambda^{k-i} \phi_i, \tag{4.2}$$

where n is the trial number and k is the step number. The parameter vector θ_n is held constant throughout trial n, and is updated only at the end of a trial.

Table 4.1 Notation used in the discussion of the TD(λ) learning rule.

x, y, z	states of the Markov chain
x_i	the state visited at time i
R_i	the reward associated with the experienced transition from state x_i to state x_{i+1}. $R_i = R(x_i, x_{i+1})$.
$\bar{R}$	the vector of expected transition rewards. $\bar{R}_x = \sum_{y \in \mathcal{X}} P(x,y)R(x,y)$. $\bar{R}_i = \sum_{y \in \mathcal{X}} P(x_i,y)R(x_i,y)$.
S	the vector of starting probabilities
V	the true value function
ϕ_x	the feature vector representation for state x
ϕ_i	the feature vector for state x_i. $\phi_i = \phi_{x_i}$.
Φ	the matrix of state representations. Row x of Φ is ϕ_x.
π_x	the proportion of time that the Markov chain is expected to spend in state x
Π	the diagonal matrix diag(π)
θ^*	the true value function parameter vector
θ_i	the estimate of θ^* at time i
$\hat{V}_i(x)$	the estimated value of state x using parameter vector θ_i
$\alpha_{\eta(x_i)}$	the learning rate used to update the value of θ_i
$\eta(x_i)$	the number of transitions from state x_i.

The theorem of Dayan and Sejnowski [23] showing convergence with probability one can be restated using our notation as follows:

Theorem 4.1 (Dayan and Sejnowski) *For an absorbing Markov chain with state set $\mathcal{X}$ consisting of the start states, $\mathcal{S}$, the terminal (or absorbing) states, $\mathcal{T}$, and the non-terminal states, $\mathcal{N}$, if (1) parameter updates are performed at the end of each trial; (2) $\mathcal{S}$ is such that there are no inaccessible states; (3) $R(x,y) = 0$ whenever both $x, y \in \mathcal{N}$; (4) $R(x,y) = 0$ whenever both $x, y \in \mathcal{T}$; (5) $R(x,y)$ is finite whenever $x \in \mathcal{N}$ and $y \in \mathcal{T}$; (6) the set of state representations $\{\phi_x \mid x \in \mathcal{N}\}$ is linearly independent; (7) $\phi_x = 0$ for all $y \in \mathcal{T}$; (8) $m = |\mathcal{N}|$ features are used to construct the state representations; (9) $\gamma = 1$; (10) $\sum_{k=1}^{\infty} \alpha_k = \infty$; and (11) $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$, then for any initial parameter vector, θ_0, the estimated value function, $\hat{V}_n(x)$, converges with probability one to the true value function, $V(x)$, as the number of trials, n, approaches infinity.*

This theorem obviously has very restricted applicability. Less restricted convergence theorems have been obtained for the TD(0) learning rule by considering it as a special case of the Q-learning algorithm. If either the policy-based or the optimizing Q-learning rules given in Equations (2.21) and (2.22) are applied to an MDP with a fixed policy U, the learning rule that results is

$$Q_{k+1}(x_k, U(x_k)) = Q_k(x_k, U(x_k)) +$$

```
Select $\theta_0$.
Set $k = 0$.
for $n = 0$ to $\infty$ {
    Choose a start state $x_k$ according to the start-state probabilities given by $S$.
    Set $\Delta_n = 0$.
    while $x_k$ is not an absorbing state {
        Take a random step from $x_k$ to $x_{k+1}$ according to the state transition probabilities.
        Set $\Delta_n = \Delta_n + \Delta\theta_k$, where $\Delta\theta_k$ is given by Equation (4.2).
        $k = k + 1$.
    }
    Update the parameters at the end of trial number $n$: $\theta_{n+1} = \theta_n + \alpha_n \Delta_n$.
}
```

Figure 4.1 Trial-based TD(λ) for absorbing Markov chains. A trial is a sequence of states that starts at some start state, follows the Markov chain as it makes transitions, and ends at an absorbing state. The variable n counts the number of trials. The variable k counts the number of steps within a trial. The parameter vector θ is updated only at the end of a trial.

$$\alpha_k(x_k, U(x_k)) \Big[\; R(x_k, x_{k+1}, U(x_k)) + \gamma Q_k(x_{k+1}, U(x_{k+1})) - Q_k(x_k, U(x_k)) \; \Big], \tag{4.3}$$

where Q_{k+1} is the estimate at time $k+1$ of Q_U. But since $Q_U(x, U(x)) = V_U(x)$, Equation (4.3) can be rewritten more clearly as

$$V_{k+1}(x_k) = V_k(x_k) + \alpha_k(x_k) \Big[\; R(x_k, x_{k+1}) + \gamma V_k(x_{k+1}) - V_k(x_k) \; \Big], \tag{4.4}$$

where $\alpha_k(x_k)$ is the learning-rate parameter at time k for state x_k. We use $R(x_k, x_{k+1})$ instead of $R(x_k, x_{k+1}, U(x_k))$ because there is no choice of action. The action is always determined by the fixed policy, U.

Watkins and Dayan [93] and Tsitsiklis [91] note that since the TD(0) learning rule is a special case of Q-learning, their probability one convergence proofs for Q-learning can be used to show that *on-line* use of the TD(0) learning rule with a lookup-table function approximator will converge with probability one. See Theorem 2.2.

4.3 Probability 1 convergence of on-line TD(0)

Here we show, as a corollary to Theorem 2.2, that the on-line use of the TD(0) learning rule with a linear function approximator and linearly independent state representations will also converge with probability one.

Theorem 4.2 (Convergence of linear TD(0)) *If the learning rates $\alpha_k(x)$ are random numbers such that (1) $\sum_{k=1}^{\infty} \alpha_k(x) = \infty$, with probability one; (2) $\sum_{k=1}^{\infty} \alpha_k(x)^2 < \infty$, with probability one; (3) the set of state representations $\{\phi_x \mid x \in \mathcal{N}\}$ is linearly independent; (4) $\phi_x = 0$ for all $y \in \mathcal{T}$; and (5) $m = |\mathcal{N}|$ features are used to construct the state representations; then $V_k(x)$ updated using the parameterized TD(0) learning rule converges to $V(x)$, with probability one, for every state x, in each of the following cases: (a) if $\gamma < 1$; or (b) if $\gamma = 1$ and the chosen policy U is proper.*

Proof: The lookup-table version of the TD(0) learning rule can be represented in vector form as

$$V_{k+1} = V_k + \alpha_k \left[\ R_k + \gamma \theta_k' \phi_{k+1} - \theta_k' \phi_k \ \right] e_k, \tag{4.5}$$

where $e_k = 0$ if x_k is an absorbing state, otherwise e_k is the standard basis vector corresponding to state x_k. Now, if row x of the matrix Φ consists of the representation for state x for all non-terminal states x, then the update rule can be rewritten for a linear approximator as

$$\Phi\theta_{k+1} = \Phi\theta_k + \alpha_k \left[\ R_k + \gamma \theta_k' \phi_{k+1} - \theta_k' \phi_k \ \right] e_k, \tag{4.6}$$

where $V_k = \Phi\theta_k$. Multiplying both sides of the equation by $(\Phi'\Phi)^{-1}\Phi'$, we get

$$\theta_{k+1} = \theta_k + \alpha_k \left[\ R_k + \gamma \theta_k' \phi_{k+1} - \theta_k' \phi_k \ \right] (\Phi'\Phi)^{-1}\Phi' e_k. \tag{4.7}$$

But by the definitions of e_k and Φ, $\Phi' e_k = \phi_k$. Therefore the update now looks like

$$\theta_{k+1} = \theta_k + \alpha_k \left[\ R_k + \gamma \theta_k' \phi_{k+1} - \theta_k' \phi_k \ \right] (\Phi'\Phi)^{-1}\phi_k. \tag{4.8}$$

We can scale the update direction by some positive definite matrix without disturbing convergence, so, scaling by the positive definite matrix $(\Phi'\Phi)$, the update rule finally becomes

$$\begin{aligned} \theta_{k+1} &= \theta_k + \alpha_k \left[\ R_k + \gamma \theta_k' \phi_{k+1} - \theta_k' \phi_k \ \right] (\Phi'\Phi)(\Phi'\Phi)^{-1}\phi_k & (4.9) \\ &= \theta_k + \alpha_k \left[\ R_k + \gamma \theta_k' \phi_{k+1} - \theta_k' \phi_k \ \right] \phi_k, & (4.10) \end{aligned}$$

which is the TD(0) learning rule for linear approximators.
Q.E.D.

4.4 The Normalized TD(λ) learning rule

As with other stochastic approximation algorithms, the size of the input vectors can cause instabilities in the TD(λ) learning rule until the size of the learning rate, α, is reduced to a small enough value. By that time, however, the convergence rate may be unacceptably slow. This section describes the Normalized TD(λ) (NTD(λ)) learning rule, and proves probability one convergence of NTD(0) with a linear function approximator when applied on-line.

The NTD(λ) learning rule for linear function approximators is

$$\theta_{k+1} = \theta_k + \alpha_k(x_k) \Big[\ R_k + \gamma \theta_k' \phi_{k+1} - \theta_k' \phi_k \ \Big] \sum_{i=1}^{k} \lambda^{k-i} \frac{\phi_i}{\epsilon + \phi_i' \phi_i}, \tag{4.11}$$

where ϵ is some small, positive number. If we know that all of the state representations are non-zero, then ϵ could be set to zero. The normalization does not change the *direction* of the updates, it merely bounds the *size* of the updates, reducing the chance for unstable behavior.

Theorem 4.3 (Convergence of linear NTD(0)) *If the learning rates $\alpha_k(x)$ are random numbers such that (1) $\sum_{k=1}^{\infty} \alpha_k(x) = \infty$, with probability one; (2) $\sum_{k=1}^{\infty} \alpha_k(x)^2 < \infty$, with probability one; (3) the set of state representations $\{\phi_x \mid x \in \mathcal{N}\}$ is linearly independent; (4) $\phi_x = 0$ for all $y \in \mathcal{T}$; (5) $m = |\mathcal{N}|$ features are used to construct the state representations; then $V_k(x)$ updated using the NTD(0) learning rule for linear function approximators (Equation (4.11)) converges to $V(x)$, with probability one, for every state x, in each of the following cases: (a) if $\gamma < 1$; or (b) if $\gamma = 1$, and the chosen policy U is proper.*

Proof: This theorem follows directly from Theorem 4.2. The scaling by $1/(\epsilon + \phi_k' \phi_k)$ serves, in effect, to replace the learning rates $\alpha_k(x)$ by the scaled learning rates

$$\beta_k(x) = \frac{\alpha_k(x)}{\epsilon + \phi_k' \phi_k}.$$

It is readily verified that the learning rates $\beta_k(x)$ satisfy conditions (1) and (2) as long as the $\alpha_k(x)$ satisfy conditions (1) and (2).
Q.E.D.

4.5 A Least Squares approach to TD learning

Stochastic approximation is a rather inefficient method of parameter estimation, since it uses each piece of information only once. The least squares approach is more efficient since it uses all of the information it has seen to form the latest estimate. This section describes a derivation of a TD learning rule based on least squares techniques.

4.5.1 Linear Least Squares function approximation

This section reviews the basics of linear least squares function approximation, including instrumental variable methods. This background material leads in the next section to an improved TD learning algorithm. Table 4.2 summarizes the notation we will use to describe the algorithms we discuss. The goal of linear least squares function approximation is to approximate some function $\Psi : \mathbb{R}^n \longrightarrow \mathbb{R}$ with a linear approximation, given samples of observed inputs $\omega_k \in \mathbb{R}^n$ and the corresponding observed outputs $\psi_k \in \mathbb{R}$. We will assume throughout this chapter that Ψ is linear. If the input observations are not corrupted by noise, then we have the situation where

$$\begin{aligned}\psi_k &= \Psi(\omega_k) + \eta_k \\ &= \omega_k' \theta^* + \eta_k. \end{aligned} \tag{4.12}$$

θ^* is the vector of true (but unknown) function parameters, and η_k is the output observation noise.

Table 4.2 Notation used in the discussion of Least Squares algorithms.

Ψ	$\Psi : \mathbb{R}^n \longrightarrow \mathbb{R}$, the linear function to be approximated.
ω_k	$\omega_k \in \mathbb{R}^n$, the observed input at time k
ψ_k	$\psi_k \in \mathbb{R}$, the observed output at time k
η_k	$\eta_k \in \mathbb{R}$, the observed output noise at time k
$\hat{\omega}_k$	$\hat{\omega}_k = \omega_k + \zeta_k$, the noisy input observed at time k
ζ_k	$\zeta_k \in \mathbb{R}^n$, the input noise at time k
$\mathrm{Cor}(x, y)$	$\mathrm{Cor}(x, y) = \mathcal{E}\{xy'\}$, the correlation matrix for random variables x and y
ρ_k	$\rho_k \in \mathbb{R}^n$, the instrumental variable observed at time k

Given Equation (4.12), the least squares approximation to θ^* at time k is the vector θ_k that minimizes the quadratic objective function

$$\mathcal{J}_k = \frac{1}{k}\sum_{i=1}^{k} \left[\psi_i - \omega_i'\theta_k\right]^2. \tag{4.13}$$

Taking the partial derivative of $\mathcal{J}_k$ with respect to θ_k, setting this equal to zero and solving for the minimizing θ_k gives us the kth estimate for θ^*,

$$\theta_k = \left[\frac{1}{k}\sum_{i=1}^{k}\omega_i\omega_i'\right]^{-1}\left[\frac{1}{k}\sum_{i=1}^{k}\omega_i\psi_i\right]. \tag{4.14}$$

Lemma (4.1) gives a set of conditions under which we can expect θ_k as defined by Equation (4.14) to converge in probability to θ^*. Appendix C.1 gives a proof based on one provided by Young [104].

Lemma 4.1 *If the correlation matrix $Cor(\omega, \omega)$ is nonsingular and finite, and the output observation noise η_i is uncorrelated with the input observations ω_i, then θ_k as defined by Equation (4.14) converges in probability to θ^*.*

Equation (4.12) models the situation where observation errors occur only on the output. In the more general case, however, the input observations are also noisy. Instead of being able to observe ω_k directly, we can only observe $\hat{\omega}_k = \omega_k + \zeta_k$, where ζ_k

is the input observation noise vector at time k. This is known as an *errors-in-variables* situation [104]. Equation (4.15) models the errors-in-variables situation.

$$\begin{aligned}\psi_k &= \Psi(\omega_k) + \eta_k \\ &= \Psi(\hat{\omega}_k - \zeta_k) + \eta_k \\ &= \hat{\omega}_k'\theta^* - \zeta_k'\theta^* + \eta_k. \end{aligned} \tag{4.15}$$

The problem with errors-in-variables is that we cannot use $\hat{\omega}_k$ instead of ω_k in Equation (4.14) without violating the conditions of Lemma (4.1). Substituting $\hat{\omega}_k$ directly for ω_k in Equation (4.14) has the effect of introducing noise that is *dependent* upon the current state. This introduces a bias, and θ_k no longer converges to θ^*. One way around this problem is to introduce *instrumental variables* [55, 79, 104]. An instrumental variable, ρ_k, is a vector that is correlated with the true input variables, ω_k, but which is uncorrelated with the observation noise, ζ_k. (Instrumental variables are difficult to obtain in most cases, but we will see that they are very easily obtained for the application of Least Squares techniques to Temporal Difference estimation.) Equation (4.16) is a modification of Equation (4.14) that uses the instrumental variables and the noisy inputs.

$$\theta_k = \left[\frac{1}{k}\sum_{i=1}^{k} \rho_i \hat{\omega}_i'\right]^{-1} \left[\frac{1}{k}\sum_{i=1}^{k} \rho_i \psi_i\right]. \tag{4.16}$$

Lemma (4.2) gives a set of conditions under which the introduction of instrumental variables solves the errors-in-variables problem. Appendix C.1 gives a proof based on one provided by Young [104].

Lemma 4.2 *If the correlation matrix $Cor(\rho, \omega)$ is nonsingular and finite, the correlation matrix $Cor(\rho, \zeta) = 0$, and the output observation noise η_k is uncorrelated with the instrumental variables ρ_k, then θ_k as defined by Equation (4.16) converges in probability to θ^*.*

4.5.2 Algorithm LS TD

This section shows how to use the instrumental variables method to derive algorithm Least Squares TD (LS TD), a least squares version of the TD learning rule. The linear TD learning rule addresses the problem of finding a parameter vector, θ^*, that allows us to compute the value of a state x as $V(x) = \phi_x'\theta^*$. Remember that the value function satisfies the following recursion

$$\begin{aligned}V(x) &= \sum_{y\in\mathcal{X}} P(x,y)[R(x,y) + \gamma V(y)] \qquad (4.17)\\ &= \sum_{y\in\mathcal{X}} P(x,y)R(x,y) + \gamma \sum_{y\in\mathcal{X}} P(x,y)V(y)\end{aligned}$$

$$= \bar{R}_x + \gamma \sum_{y \in \mathcal{X}} P(x,y)V(y) \tag{4.18}$$

We can rewrite this in the form of Section 4.5.1 as

$$\begin{aligned} \bar{R}_x &= V(x) - \gamma \sum_{y \in \mathcal{X}} P(x,y)V(y) \\ &= \phi_x'\theta^* - \gamma \sum_{y \in \mathcal{X}} P(x,y)\phi_y'\theta^* \\ &= (\phi_x - \gamma \sum_{y \in \mathcal{X}} P(x,y)\phi_y)'\theta^*, \end{aligned} \tag{4.19}$$

for every state $x \in \mathcal{X}$. Now we have the same kind of problem that we considered in Section 4.5.1. The scalar output, $\bar{R}_x$, is the inner product of the input vector, $(\phi_x - \gamma \sum_{y \in \mathcal{X}} P(x,y)\phi_y)$, and the vector of true function parameters, θ^*.

Looking at the problem in this way allows us to solve a relatively simple problem. Suppose that we know everything about the Markov chain except R, the matrix of transition costs. We know P, the matrix of state transition probabilities; we know ϕ_x, the state representation for every state x; and we know at every time step the state of the Markov chain. At every time step k we observe the current state, x_k, and the reward received on transition to state x_{k+1}, R_k. We then have an equation similar to Equation (4.12)

$$R_k = (\phi_k - \gamma \sum_{y \in \mathcal{X}} P(x_k,y)\phi_y)'\theta^* + (R_k - \bar{R}_k). \tag{4.20}$$

$(R_k - \bar{R}_k)$ corresponds to the noise term η_k in Equation (4.12). Lemma (4.3) establishes that the noise term $\eta_k = R_k - \bar{R}_k$ is zero-mean and uncorrelated with the input vector $\omega_k = \phi_k - \gamma \sum_{y \in \mathcal{X}} P(x_k,y)\phi_y$. Appendix C.3 gives a proof. Therefore, if $\mathrm{Cor}(\omega,\omega)$ is nonsingular and finite, then by Lemma (4.1) we can use the algorithm given by Equation (4.14) to find θ^*.

Lemma 4.3 *For any Markov chain, if x and y are states such that $P(x,y) > 0$, with $\eta_{xy} = R(x,y) - \bar{R}_x$ and $\omega_x = (\phi_x - \gamma \sum_{y \in \mathcal{X}} P(x,y)\phi_y)$, then $\mathcal{E}\{\eta\} = 0$, and $Cor(\omega,\eta) = 0$.*

In the general case, however, we know neither the state transition probabilities nor the state representations, nor can we directly measure the state of the Markov chain. All that is known at each time step are ϕ_k and ϕ_{k+1}, the representations of states at either end of a state transition, and R_k, the corresponding reward. Instrumental variable methods allow us to solve this problem, too. Take $\hat{\omega}_k = \phi_k - \gamma\phi_{k+1}$, and $\zeta_k = \gamma \sum_{y \in \mathcal{X}} P(x_k,y)\phi_y - \gamma\phi_{k+1}$. Now we are in an errors-in-variables situation, as demonstrated in the following equation.

$$\hat{\omega}_k = \phi_k - \gamma\phi_{k+1}$$

$$= (\phi_k - \gamma \sum_{y \in \mathcal{X}} P(x_k, y)\phi_y) + (\gamma \sum_{y \in \mathcal{X}} P(x_k, y)\phi_y - \gamma\phi_{k+1})$$
$$= \omega_k + \zeta_k,$$

where $\omega_k = \phi_k - \gamma \sum_{y \in \mathcal{X}} P(x_k, y)\phi_y$, and $\zeta_k = \gamma \sum_{y \in \mathcal{X}} P(x_k, y)\phi_y - \gamma\phi_{k+1}$.

Section 4.5.1 shows how to use instrumental variables to avoid the asymptotic bias introduced by errors-in-variables problems. Lemma (4.4) shows that the instrumental variable $\rho_k = \phi_k$ is uncorrelated with the input observation noise, ζ_k, as defined above. Appendix C.3 gives a proof. Lemma (4.4) establishes that this choice of instrumental variables allows us to discount the effect of the input noise on the parameter estimates. It still remains to show in Section 4.5.3 under what conditions we can expect to $\mathrm{Cor}(\rho, \omega)$ to be finite and nonsingular, which completes the proof that algorithm LS TD converges to θ^*.

Lemma 4.4 *For any Markov chain, if (1) x and y are states such that $P(x, y) > 0$; (2) $\zeta_{xy} = \gamma \sum_{z \in \mathcal{X}} P(x, z)\phi_z - \gamma\phi_y$; (3) $\eta_{xy} = R(x, y) - \bar{R}_x$; and (4) $\rho_x = \phi_x$, then (1) $Cor(\rho, \eta) = 0$; and (2) $Cor(\rho, \zeta) = 0$.*

Using ϕ_x as our candidate instrumental variable, we rewrite Equation (4.16) to explicitly give the LS TD algorithm.

$$\theta_k = \left[\ \tfrac{1}{k} \textstyle\sum_{i=1}^{k} \phi_i(\phi_i - \gamma\phi_{i+1})' \ \right]^{-1} \left[\ \tfrac{1}{k} \textstyle\sum_{i=1}^{k} \phi_i R_i \ \right]. \tag{4.21}$$

Figure 4.2 shows how Equation (4.16) can be used as part of a trial-based algorithm to find the value function for an absorbing Markov chain. Figure 4.3 shows how Equation (4.16) can be used as part of an algorithm to find the value function for an ergodic Markov chain. Learning takes place on-line in both algorithms, with parameter updates after every state transition. The parameter vector θ_k will not be well defined while k is small, since the matrix $\left[\ \frac{1}{k} \sum_{i=1}^{k} \phi_i(\phi_i - \gamma\phi_{i+1})' \ \right]$ will not be invertible. The next section discusses the convergence of LS TD as applied to absorbing and ergodic Markov chains.

4.5.3 Convergence of algorithm LS TD

In this section we consider the asymptotic performance of algorithm LS TD when used on-line to approximate the value functions of absorbing and ergodic Markov chains. Lemma (4.5) starts the analysis by expressing $\theta_{\text{LSTD}} \overset{\text{def}}{=} \lim_{k\to\infty} \theta_k$, the limiting estimate found by algorithm LS TD for θ^*, in a convenient form. Appendix C.3 gives a proof.

Lemma 4.5 *For any Markov chain, when (1) θ_k is found using algorithm LS TD; (2) each state $x \in \mathcal{X}$ is visited infinitely often; (3) each state $x \in \mathcal{X}$ is visited in the long run with probability 1 in proportion π_x; and (4) $[\Phi'\Pi(I - \gamma P)\Phi]$ is invertible, then $\theta_{\text{LSTD}} = [\Phi'\Pi(I - \gamma P)\Phi]^{-1} \left[\Phi'\Pi\bar{R}\right]$ with probability 1.*

```
1  Set $k = 0$.
2  repeat forever {
3      Set $x_k$ to be a start state selected according to the prob-
       abilities given by $S$.
4      while $x_k$ is not an absorbing state {
5          Take a random step from $x_k$ to $x_{k+1}$ according to the
           state transition probabilities.
6          Use Equation (4.21) to define $\theta_k$.
7          $k = k + 1$.
8      }
9  }
```

Figure 4.2 Trial-based LS TD for absorbing Markov chains. A trial is a sequence of states that starts at some start state, follows the Markov chain as it makes transitions, and ends at an absorbing state.

```
1  Set $k = 0$.
2  Select an arbitrary initial state, $x_0$.
3  repeat forever {
4      Take a random step from $x_k$ to $x_{k+1}$ according to the
       state transition probabilities.
5      Use Equation (4.21) to define $\theta_k$.
6      $k = k + 1$.
7  }
```

Figure 4.3 LS TD for ergodic Markov chains.

The key to using Lemma (4.5) lies in the definition of π. As defined in Table 4.1, π_x is the proportion of time that the Markov chain is expected to spend over the long term in state x. Equivalently, π_x is the expected proportion of state transitions that take the Markov chain out of state x. For an ergodic Markov chain, π_x is the invariant or steady-state distribution associated with the stochastic matrix P [44]. For an absorbing Markov chain, π_x is the expected number of visits to state x during one transition sequence from a start state to a goal state [44]. Since there are no state transitions out of goal states, $\pi_x = 0$ for all goal states. These definitions prepare the way for the following two theorems. Theorem 4.4 gives conditions under which LS TD as used in Figure 4.2 will cause θ_{LSTD} to converge with probability 1 to θ^* when applied to an absorbing Markov chain. Theorem 4.5 gives conditions under which LS TD as used in Figure 4.3 will cause θ_{LSTD} to converge with probability 1 to θ^* when applied to an ergodic Markov chain. Section 4.5.4 gives a proof for Theorem 4.4, while Section 4.5.5 gives a proof for Theorem 4.5.

Theorem 4.4 (Convergence of LS TD for absorbing Markov chains) *When using LS TD as described in Figure 4.2 to estimate the value function for an absorbing Markov chain, if (1) S is such that there are no inaccessible states; (2) $R(x,y)$ is finite whenever both $x,y \in \mathcal{N}$; (3) $R(x,y) = 0$ whenever both $x,y \in \mathcal{T}$; (4) $R(x,y)$ is finite whenever $x \in \mathcal{N}$ and $y \in \mathcal{T}$; (5) the set of state representations $\{\phi_x \mid x \in \mathcal{N}\}$ is linearly independent; (6) $\phi_x = 0$ for all $y \in \mathcal{T}$; (7) $m = |\mathcal{N}|$ features are used to construct the state representations; and (8) $0 \leq \gamma \leq 1$, then θ^* is finite and the asymptotic parameter estimate found by algorithm LS TD, θ_{LSTD}, converges with probability 1 to θ^* as the number of trials (and state transitions) approaches infinity.*

Different conditions are required in the absorbing and ergodic chain cases in order to meet the conditions of Lemma (4.5). The conditions required in Theorem 4.4 are generalizations of the conditions required in Theorem 4.1. The conditions for Theorem 4.5 are much less restrictive, though the discount factor γ must be less than 1 to ensure that the value function is finite.

Theorem 4.5 (Convergence of LS TD for ergodic Markov chains) *When using LS TD as described in Figure 4.3 to estimate the value function for an ergodic Markov chain, if (1) the set of state representations $\{\phi_x \mid x \in \mathcal{X}\}$ is linearly independent; (2) $N = |\mathcal{X}|$ features are used to construct the state representations; (3) $0 \leq \gamma < 1$; and (4) $R(x,y)$ is finite for all $x,y \in \mathcal{X}$, then θ^* is finite and the asymptotic parameter estimate found by algorithm LS TD, θ_{LSTD}, converges with probability 1 to θ^* as the number of state transitions approaches infinity.*

Theorems 4.4 and 4.5 provide convergence assurances for LS TD similar to those provided by Tsitsiklis [91] or Watkins and Dayan [93] for the convergence of TD(0) using a lookup-table function approximator.

4.5.4 Proof of Theorem 4.4

Proof: Condition (1) implies that, with probability one, as the total number of state transitions approaches infinity, the number of times each state $x \in \mathcal{X}$ is visited approaches infinity. Since this is an absorbing chain, we have with probability 1 that the proportion of time the states are visited approaches π as the number of trials approaches infinity. Therefore, by Lemma (4.5), we know that with probability 1

$$\theta_{\text{LSTD}} = \left[\Phi'\Pi(I-\gamma P)\Phi\right]^{-1}\left[\Phi'\Pi\bar{R}\right],$$

assuming that the inverse exists.

Conditions (5), (6), and (7) imply that Φ has rank m, with row x of Φ consisting of all zeros for all $x \in \mathcal{T}$. Condition (1) implies that Π has rank m. Row x of Π consists of all zeros, for all $x \in \mathcal{T}$. The state representation matrix Φ has the property that if all rows corresponding to goal states are removed, the resulting submatrix is of dimensions $(m \times m)$, and has rank m. Call this submatrix A. Matrix Π has the property that if all rows and columns corresponding to goal states are removed, the resulting submatrix is of dimensions $(m \times m)$, and has rank m. Call this submatrix

B. The matrix $(I-\gamma P)$ has the property that if all rows and columns corresponding to goal states are removed, the resulting submatrix is of dimensions $(m \times m)$, and has rank m [44]. Call this submatrix C. It can be verified directly by performing the multiplicatons that $[\Phi'\Pi(I-\gamma P)\Phi] = [A'BCA]$. Therefore, $[\Phi'\Pi(I-\gamma P)\Phi]$ is of dimensions $(m \times m)$, and has rank m. Thus, it is invertible.

Now, Equation (4.19) can be rewritten using matrix notation as

$$\bar{R} = (I-\gamma P)\Phi\theta^* \tag{4.22}$$

This, together with conditions (2), (3), (4), and (8) implies that θ^* is finite. Finally, substitutiong Equation (4.22) into the expression for θ_{LSTD} gives us

$$\begin{aligned}\theta_{\text{LSTD}} &= [\Phi'\Pi(I-\gamma P)\Phi]^{-1}\,[\Phi'\Pi(I-\gamma P)\Phi]\,\theta^* \\ &= \theta^*.\end{aligned}$$

Thus, θ_{LSTD} converges to θ^* with probability 1.
Q.E.D.

4.5.5 Proof of Theorem 4.5

Proof: Since this is an ergodic chain, as k approaches infinity we have with probability 1 that the number of times each of the states $x \in \mathcal{X}$ is visited approaches infinity. We also have with probability 1 that the states are visited in the long run in proportion π. Ergodicity implies that $\pi_x > 0$ for all $x \in \mathcal{X}$. Therefore, Π is invertible. Condition (3) implies that $(I-\gamma P)$ is invertible. Conditions (1) and (2) imply that Φ is invertible. Therefore, by Lemma (4.5), we know that with probability 1

$$\theta_{\text{LSTD}} = [\Phi'\Pi(I-\gamma P)\Phi]^{-1}\left[\Phi'\Pi\bar{R}\right].$$

Conditions (3) and (4), together with Equation (4.22) imply that θ^* is finite. And, as above, substituting Equation (4.22) into the expression for θ_{LSTD} gives us

$$\begin{aligned}\theta_{\text{LSTD}} &= [\Phi'\Pi(I-\gamma P)\Phi]^{-1}\,[\Phi'\Pi(I-\gamma P)\Phi]\,\theta^* \\ &= \theta^*.\end{aligned}$$

Thus, θ_{LSTD} converges to θ^* with probability 1.
Q.E.D.

4.5.6 Algorithm RLS TD

Algorithm LS TD requires the computation of a matrix inverse at each time step. This means that LS TD has a computational complexity of $O(m^3)$, assuming that the state representations are of length m. We can use Recursive Least Squares (RLS) techniques [30, 55, 104] to derive a modified algorithm, Recursive Least Squares TD (RLS TD), with computational complexity of order $O(m^2)$. Equations (4.23) specify algorithm RLS TD. Notice that Equation (4.23c) is the TD(0) learning rule for linear function approximators, except that the scalar learning rate has been replaced by a gain matrix.

$$e_k = R_k - (\phi_k - \gamma\phi_{k+1})'\theta_{k-1} \tag{4.23a}$$

$$C_k = C_{k-1} - \frac{C_{k-1}\phi_k(\phi_k - \gamma\phi_{k+1})'C_{k-1}}{1 + (\phi_k - \gamma\phi_{k+1})'C_{k-1}\phi_k} \tag{4.23b}$$

$$\theta_k = \theta_{k-1} + \frac{C_{k-1}}{1 + (\phi_k - \gamma\phi_{k+1})'C_{k-1}\phi_k} e_k\phi_k \tag{4.23c}$$

The user of a RLS algorithm must specify θ_0 and C_0. C_k is the k^{th} sample estimate of $\frac{1}{k}\text{Cor}(\rho, \hat{\omega})^{-1}$, where ρ and $\hat{\omega}$ are defined as in Section 4.5.2. C_0^{-1} is typically chosen to be a diagonal matrix of the form βI, where β is some large positive constant. This ensures that C_0, the initial guess at the correlation matrix, is approximately 0, but is invertible and symmetric positive definite.

The convergence of algorithm RLS TD requires the same conditions as algorithm LS TD, plus one more. This is condition A.1, or equivalently, condition A.2, below:

Condition A.1: $\left[C_0^{-1} + \sum_{i=1}^k \rho_i\hat{\omega}_i'\right]$ is non-singular for all k.

Condition A.2: $[1 + \hat{\omega}_k' C_{k-1}\rho_k] \neq 0$ for all times k.

Assuming that the conditions A.1 and A.2 are maintained, $C_k = \left[C_0^{-1} + \sum_{i=1}^k \rho_i\hat{\omega}_i'\right]^{-1}$ and

$$\begin{aligned}\theta_k &= \left[C_0^{-1} + \sum_{i=1}^k \rho_i\hat{\omega}_i'\right]^{-1}\left[C_0^{-1}\theta_0 + \sum_{i=1}^k \rho_i\psi_i\right] \\ &= \left[\frac{1}{k}C_0^{-1} + \frac{1}{k}\sum_{i=1}^k \rho_i\hat{\omega}_i'\right]^{-1}\left[\frac{1}{k}C_0^{-1}\theta_0 + \frac{1}{k}\sum_{i=1}^k \rho_i\psi_i\right].\end{aligned} \tag{4.24}$$

If the conditions A.1 and A.2 are not met at some time k_0, then all computations made thereafter will be polluted by the indeterminate or infinite values produced at time k_0. The non-recursive algorithm LS TD does not have this problem, because the computations made at any time step do not depend directly on the results of computations made at earlier time steps.

4.5.7 Dependent or extraneous features

The value function for a Markov chain satisfies the equation

$$V = \left[\ I - \gamma P\ \right]^{-1} \bar{R}.$$

When using a linear function approximator, this means that θ must satisfy the linear equation

$$\Phi\theta = \left[\ I - \gamma P\ \right]^{-1} \bar{R}. \tag{4.25}$$

In this section, the rows of Φ will consist only of the representations for the non-terminal states, and V will only contain the values for the non-terminal states. This is not essential, but it makes the discussion much simpler. Let $n = |\mathcal{N}|$ be the number of non-terminal states in the Markov chain. Matrix Φ is of dimensions $n \times m$, where m is the length of the state representations.

Now, suppose that $\text{rank}(\Phi) = m < n$. Dayan [22] shows that the trial-based TD(λ) given in Figure 4.1 converges in this case to $[\Phi'\Pi(I - \gamma P)\Phi]^{-1} \left[\Phi'\Pi\bar{R}\right]$ for $\lambda = 0$. This is same result we achieved in Lemma (4.5), since $m = \text{rank}(\Phi)$ if and only if $[\Phi'\Pi(I - \gamma P)\Phi]$ is invertible[1] The proofs of Theorems 4.4 and 4.5 show convergence of θ_{LSTD} to $[\Phi'\Pi(I - \gamma P)\Phi]^{-1} \left[\Phi'\Pi\bar{R}\right]$ as a preliminary result. Thus, θ_{LSTD} will converge for both absorbing and ergodic chains as long as the assumptions of Lemma (4.5) are satisfied.

Suppose, on the other hand, that $\text{rank}(\Phi) = n < m$. This means that the state representations are linearly independent, but contain extraneous features. Therefore, we are in the situation where we have more adjustable parameters in Equation (4.25) than we have constraints. In this case there are an infinite number of parameter vectors θ that satisfy Equation (4.25). The stochastic approximation algorithms TD(λ) and NTD(λ) will converge to some θ that satisfies Equation (4.25). Which one they find depends on the order in which the states are visited. LS TD will not converge, since $\left[\ \frac{1}{k}\sum_{i=1}^{k} \phi_i(\phi_i - \gamma\phi_{i+1})'\ \right]$ will never be invertible. However, RLS TD will converge to some θ that satisfies Equation (4.25). In this case, too, the θ to which the algorithm converges depends on the order in which the states are visited.

4.6 A related algorithm of Werbos

Werbos [96] proposed the following algorithm as a linear version of his Heuristic Dynamic Programming (HDP) algorithm [57, 94, 95, 97].

$$\theta_k = \left[\frac{1}{k}\sum_{i=1}^{k} \phi_i\phi_i'\right]^{-1} \left[\frac{1}{k}\sum_{i=1}^{k} \phi_i R_i + \gamma\frac{1}{k}\sum_{i=1}^{k} \phi_i\phi_{i+1}'\theta_{k-1}\right]. \tag{4.26}$$

Werbos' derivation is heuristic, and will not be repeated here. However, this linear HDP has obvious similarities to LS TD. Linear HDP is not as amenable to a recursive formulation as LS TD due to the presence of θ_{k-1} on the right-hand side of Equation (4.26). However, it can be written recursively as follows:

$$C_k = C_{k-1} - \frac{C_{k-1}\phi_k\phi_k'C_{k-1}}{1 + \phi_k'C_{k-1}\phi_k} \tag{4.27a}$$

$$U_k = U_{k-1} + \phi_k R_k \tag{4.27b}$$

$$W_k = W_{k-1} + \gamma\phi_k\phi_{k+1}' \tag{4.27c}$$

$$\theta_k = C_k\left[U_k + W_k\theta_{k-1}\right]. \tag{4.27d}$$

The asymptotic convergence of algorithm HDP is not as easy to analyze as it was for algorithm LS TD, again due to the presence of θ_{k-1} on the right-hand side

[1] m cannot be less than $\text{rank}(\Phi)$. If $m > \text{rank}(\Phi)$, then $[\Phi'\Pi(I - \gamma P)\Phi]$ is an $(m \times m)$ matrix with rank less than m. It is therefore not invertible.

of Equation (4.26). But, if we assume that it *has* converged to some value we can determine what that value would be. After the parameter estimate formed by HDP has converged to θ_{HDP}, Equation (4.26) becomes

$$\theta_{\text{HDP}} = \left[\frac{1}{k}\sum_{i=1}^{k}\phi_i\phi_i'\right]^{-1}\left[\frac{1}{k}\sum_{i=1}^{k}\phi_i R_i + \gamma\frac{1}{k}\sum_{i=1}^{k}\phi_i\phi_{i+1}'\theta_{\text{HDP}}\right].$$

Taking the limit as k goes to ∞ gives us

$$\begin{aligned}\theta_{\text{HDP}} &= \lim_{k\to\infty}\left[\frac{1}{k}\sum_{i=1}^{k}\phi_i\phi_i'\right]^{-1}\left[\frac{1}{k}\sum_{i=1}^{k}\phi_i R_i + \gamma\frac{1}{k}\sum_{i=1}^{k}\phi_i\phi_{i+1}'\theta_{\text{HDP}}\right]\\ &= \left[\sum_x \pi_x\phi_x\phi_x'\right]^{-1}\left[\sum_x \pi_x\phi_x R_x + \gamma\sum_x \pi_x\phi_x\sum_y P(x,y)\phi_y'\theta_{\text{HDP}}\right]\\ &= \left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi\bar{R} + \gamma\Phi'\Pi P\Phi\theta_{\text{HDP}}\right].\end{aligned}$$

Moving all instances of θ_{HDP} to the left-hand side, we get

$$\theta_{\text{HDP}} - \gamma\left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi P\Phi\right]\theta_{\text{HDP}} = \left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi\bar{R}\right],$$

or

$$\left[I - \gamma\left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi P\Phi\right]\right]\theta_{\text{HDP}} = \left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi\bar{R}\right].$$

Assuming that all of the matrix inverses we need actually exist, we then get

$$\begin{aligned}\theta_{\text{HDP}} &= \left[I - \gamma\left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi P\Phi\right]\right]^{-1}\left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi\bar{R}\right]\\ &= \left[\left[\Phi'\Pi\Phi\right]^{-1}\left[\left[\Phi'\Pi\Phi\right] - \gamma\left[\Phi'\Pi P\Phi\right]\right]\right]^{-1}\left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi\bar{R}\right]\\ &= \left[\left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi(I-\gamma P)\Phi\right]\right]^{-1}\left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi\bar{R}\right]\\ &= \left[\Phi'\Pi(I-\gamma P)\Phi\right]^{-1}\left[\Phi'\Pi\Phi\right]\left[\Phi'\Pi\Phi\right]^{-1}\left[\Phi'\Pi\bar{R}\right]\\ &= \left[\Phi'\Pi(I-\gamma P)\Phi\right]^{-1}\left[\Phi'\Pi\bar{R}\right].\end{aligned}$$

All of the necessary inverses will exist, as discussed above, under the conditions of either Theorem 4.4 or Theorem 4.5. Thus, if linear HDP converges, it converges to the same parameter vector as LS TD. Experimentally, however, linear HDP appears to be only locally convergent, even when the conditions of Theorems 4.4 or 4.5 are satisfied. The presence of θ_{k-1} on the right-hand side of Equation (4.26) means that the algorithm is sensitive to the initial parameter estimate, θ_0. If θ_0 is close enough to θ^*, then linear HDP converges, but otherwise it will not. Even when it does converge, convergence is slowed by the presence of θ_{k-1} on the right-hand side of Equation (4.26).

4.7 Choosing an algorithm

The linear TD(λ) and NTD(λ) algorithms involve costs of order $O(m)$ at each time step when measured either in terms of the number of basic computer operations, or in terms of memory requirements, where m is the length of the state representation vectors. Algorithm LS TD's costs are of order $O(m^3)$ in time and $O(m^2)$ in space at each time step, while RLS TD's are of order $O(m^2)$ in both time and space. TD(λ) and NTD(λ) are clearly superior in terms of cost per time step. However, LS TD and RLS TD are more efficient estimators in the statistical sense. They extract more of the available information from each additional piece of data. Therefore, we would expect LS TD and RLS TD to converge more rapidly than TD(λ) and NTD(λ). The use of LS TD and RLS TD is justified then, if the increased costs per time step are offset by increased convergence speed.

The performance of TD(λ) is sensitive to a number of interrelated factors that do not affect the performance of either LS TD or RLS TD. Convergence of TD(λ) can be dramatically slowed by a poor choice of the learning rate (α) and trace (λ) parameters. The algorithm can become unstable if α is too large, causing θ_k to diverge. TD(λ) is sensitive to the size of the state representation vectors. If vector ϕ_x has some large component, then the update along that dimension will also tend to be large, perhaps too large for stability. Judicious choice of α and λ can damp down the instability. Unfortunately, the price for stability may be slow convergence. The performance of TD(λ) is also sensitive to $|\theta_0 - \theta^*|$, the distance between θ^* and the initial estimate for θ^*. NTD(λ) is sensitive to these same factors, but normalization reduces the sensitivity. A second way that algorithms LS TD and RLS TD can be justified is by the fact that they are insensitive to all of these factors. Use of LS TD and RLS TD eliminates the possibility of poor performance due to unlucky choice of parameters.

The transient performance of the learning algorithm may be important. TD(λ) and NTD(λ) will remain stable (assuming that the learning rate is small enough) no matter what sequence of states is visited. This is not true for LS TD and RLS TD. If $C_k^{-1} = \left[C_0^{-1} + \sum_{i=1}^{k} \rho_i \hat{\omega}_i'\right]$ is ill-conditioned or singular for some time k, then the estimate θ_k can far from θ^*. LS TD will recover from this transient event, and is assured of converging eventually to θ^*. The version of RLS TD described in Section 4.5.6 will not recover if C_k^{-1} is singular. It may or may not recover from an ill-conditioned C_k^{-1}, depending on the machine arithmetic. However, there are well-known techniques for protecting RLS algorithms from transient instability [30].

TD(λ), NTD(λ), and RLS TD have an advantage over LS TD in the case of extraneous features, as discussed in Section 4.5.7. TD(λ), NTD(λ), and RLS TD will converge to the correct value function in this situation, while LS TD will not.

None of the factors discussed in this section makes a definitive case for one algorithm or another. The choice depends finally on the computational cost structure imposed on the user of these algorithms.

4.8 The TD error variance

One of the interesting characteristics of the TD error term,

$$e_{\text{TD}}(\theta_{k-1}) = R_k + \gamma\phi'_{k+1}\theta_{k-1} - \phi'_k\theta_{k-1},$$

is that it does not go to zero as θ_k converges to θ^*, except in the trivial case that the Markov chain is deterministic. This is readily verified by inspection of Equation (4.20). We define the *TD error variance*, σ_{TD}, of a Markov chain as

$$\begin{aligned}\sigma_{\text{TD}} &= \mathcal{E}\left\{e_{\text{TD}}(\theta^*)^2\right\} \\ &= \mathcal{E}\left\{\left[R_k + \gamma\phi'_{k+1}\theta^* - \phi'_k\theta^*\right]^2\right\} \\ &= \sum_{x\in\mathcal{X}} \pi_x \sum_{y\in\mathcal{X}} P(x,y)\left[R(x,y) + \gamma\phi'_y\theta^* - \phi'_x\theta^*\right]^2.\end{aligned}$$

σ_{TD} is the variance of the TD error term under the assumptions that θ_k has converged to θ^*, and that the states (and the corresponding TD errors) are sampled on-line by following a sample path through the Markov chain. σ_{TD} is a measure of the noise that cannot be removed from any of the TD learning rules (TD(λ), NTD(λ), LS TD, or RLS TD), even after parameter convergence. It seems reasonable to expect that experimental convergence rates depend on σ_{TD}.

Singh [77] considers a similar concept when discussing the tradeoffs between algorithms such as TD(λ) and RTDP. TD(λ)and the other algorithms discussed in this chapter perform a *sample backup*. They update the value function estimate using the information from sampled state transitions. RTDP performs a *full backup*. It uses a model to update the value function estimate using information from all possible state transitions. Singh concludes that algorithms performing sample backups will have a computational advantage over those performing full backups when the state transition probabilities for any state are concentrated in only a few next states. That is, for all states x, $P(x,y) \approx 0$ for nearly all states y. But σ_{TD}, the inherent noise in a sample backup, could still be relatively large under these conditions, depending on the form of the cost and value functions. Therefore we believe that σ_{TD} could be used to form a clearer measure of when sample backups will have an advantage over full backups.

4.9 Experiments

This section describes an experiment designed to demonstrate the advantage in convergence speed that can be gained through using least squares techniques. The experiment compares the performance of NTD(λ) with that of RLS TD in the on-line estimation of the value function for a randomly generated Markov chain. In a preliminary series of experiments, not reported here, NTD(λ) always performed at least as well as TD(λ), while showing less sensitivity to the choice of parameters, such as initial learning rate. Section 4.9.1 describes the algorithm used to set the learning rates for NTD(λ). Figure 4.4 shows the experimental results.

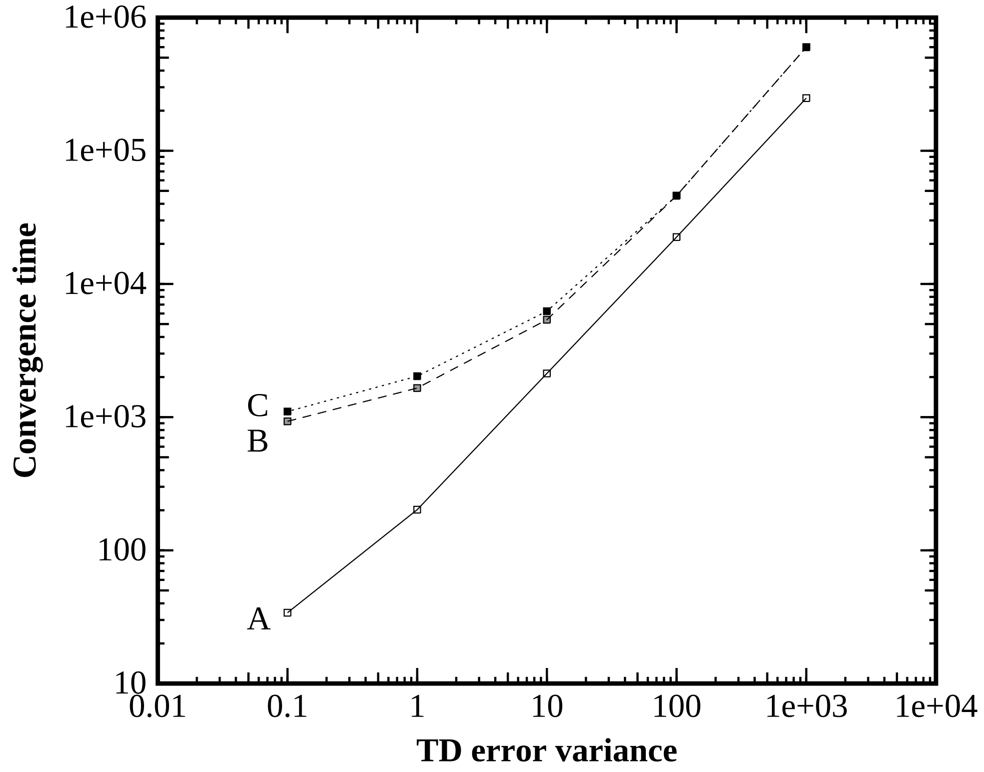


Figure 4.4 Comparison of RLS TD and NTD(λ) on a 5 state ergodic Markov chain. The x-axis measures the TD error variance of the test Markov chain, which was varied over five distinct values from $\sigma_{\mathrm{TD}} = 10^{-1}$ through $\sigma_{\mathrm{TD}} = 10^{3}$ by scaling the cost matrix, R. The y-axis measures the average convergence time over 100 training runs of on-line learning. The parameter vector is considered to have converged when the average of the error $\|\theta_k - \theta^*\|_\infty$ falls below 10^{-2} and stays below 10^{-2} thereafter. Line A shows the performance of RLS TD. Line B shows the performance of NTD(λ) where $\|\theta_0 - \theta^*\|_2 = 1$. Line C shows the performance of NTD(λ) where $\|\theta_0 - \theta^*\|_2 = 2$.

The x-axis of Figure 4.4 measures the TD error variance of the test Markov chain, which was varied over five distinct values from $\sigma_{\text{TD}} = 10^{-1}$ through $\sigma_{\text{TD}} = 10^{3}$ by scaling the cost matrix, R. The state transition probability matrix P and the state representations Φ were left unchanged. The y-axis of Figure 4.4 measures the average convergence time over 100 training runs of on-line learning. This is computed as follows. For each of the 100 training runs, the distance $\|\theta_k - \theta^*\|_\infty$ is recorded at each time step. These 100 error curves are averaged together to produce the mean error curve. Finally, the mean error curve is inspected to find the time k at which the average error falls below 10^{-2} and stays below 10^{-2} for all times thereafter.

Line A shows the performance of RLS TD. The other two lines show the performance of NTD(λ) given different initial values for θ_0. Line B shows the performance of NTD(λ) where θ_0 is chosen so that $\|\theta_0 - \theta^*\|_2 = 1$. Line C shows the performance of NTD(λ) where θ_0 is chosen so that $\|\theta_0 - \theta^*\|_2 = 2$. The performance of NTD(λ) depends the initial distance from θ^*. The conditions imposed on θ_0 are an attempt to control for this effect. The performance of NTD(λ) is also sensitive to the settings of four control parameters: λ, α_0, c, and τ. The parameters c and τ govern the evolution of the sequence of learning rates. The algorithm used to determine α_k is described in Section 4.9.1. A search for the best set of control parameters for NTD(λ) was performed at each experiment in an attempt to present NTD(λ) in the best light[2] The control parameter ϵ (see Equation (4.11)) was held constant at 1.0 for all experiments.

Figure 4.4 shows a number of things. First, RLS TD outperformed NTD(λ) at every level of σ_{TD}. RLS TD always converged at least twice as fast as NTD(λ), and did much better than that at lower levels of σ_{TD}. Next, we see that, at least for RLS TD, convergence time is a linear function of σ_{TD}. Increase σ_{TD} by a factor of 10, and the convergence time can be expected to increase by a factor of 10. This relationship is less clear for NTD(λ), though the convergence curves seem to follow the same rule for larger σ_{TD}. It appears that the effect of the initial distance from θ to θ^*, $\|\theta_0 - \theta^*\|_2$, dominates when σ_{TD} is small, but becomes less important and is finally eliminated as σ_{TD} increases.

Judging by these results, the use of RLS TD instead of TD(λ) or NTD(λ) is easily justified. RLS TD's costs per time step are an order of $n = |\mathcal{X}|$ more expensive in both time and space than the costs for TD(λ) or NTD(λ). However, RLS TD always converges significantly faster than TD(λ) or NTD(λ), and at least an order of n faster for smaller σ_{TD}. These results show TD(λ) or NTD(λ) at their best, that is, for optimal settings for λ, α_0, c, and τ. It required a very extensive parameter search to get this level of performance. It is easy to make TD(λ) or NTD(λ) look arbitrarily bad through poor selection of the algorithm parameters, and the choice of good parameters is still more a matter of luck and experience than of science. If one's objective is to find the parameters for the value function instead of comparing algorithm performance, then RLS TD has the advantage, since it has no such parameters to choose.

[2]The search for the best settings for λ, α_0, c, and τ was the limiting factor on the size of the state space for this experiment. A larger state space requires, in general, a longer convergence time.

4.9.1 Selecting learning rates

The convergence theorem for NTD(0) (Theorem 4.3) requires that there be a separate learning rate, $\alpha(x)$, for each state, and that each learning rate satisfy the Robbins and Monro [66] criteria

$$\sum_{k=1}^{\infty} \alpha_k(x) = \infty \qquad \text{and} \qquad \sum_{k=1}^{\infty} \alpha_k(x)^2 < \infty$$

with probability one, where $\alpha_k(x)$ is the learning rate for the k^{th} update of the value of state x. Instead of a separate learning rate for each state, we used a single learning rate α_k, which is decreased at every state transition. For each state there is a corresponding subsequence $\{\alpha_k\}_x$ that is used to update the value function estimate for that state. We conjecture that, if the original sequence $\{\alpha_k\}$ satisfies the Robbins and Monro criteria, then these subsequences also satisfy the criteria, with probability one. The overall convergence rate may be decreased by use of a single learning rate since each subsequence will contain fewer large learning rates.

The learning rate sequence $\{\alpha_k\}$ was generated using the "search then converge" algorithm described by Darken, Chang, and Moody [20],

$$\alpha_k = \alpha_0 \frac{1 + \frac{c}{\alpha_0}\frac{k}{\tau}}{1 + \frac{c}{\alpha_0}\frac{k}{\tau} + \tau\frac{k^2}{\tau^2}}.$$

The choice of parameters α_0, c, and τ determines the transition of the learning rate from "search mode" to "converge mode". Search mode describes the time during which $k \ll \tau$. Converge mode describes the time during which $k \gg \tau$. α_k is nearly constant in search mode, while $\alpha_k \approx \frac{c}{k}$ in converge mode. The ideal choice of learning rate parameters moves θ_k as quickly as possible into the vicinity of θ^* during search mode, and then settles into converge mode.

4.10 Conclusions

In this chapter we described three new TD learning algorithms: NTD(λ) (in Section 4.4), LS TD (in Section 4.5.2), and RLS TD (in Section 4.5.6). We also proved probability one convergence for these algorithms under appropriate conditions. These algorithms have a number of advantages over previously proposed TD learning algorithms. NTD(λ) is a normalized version of linear TD(λ). The normalization serves to reduce the algorithm's sensitivity to poorly chosen control parameters. LS TD is a Least Squares algorithm for finding the value function of a Markov chain. Although LS TD is more expensive per time step than the stochastic approximation algorithms TD(λ) and NTD(λ), it converges more rapidly. It has the further advantage that there are no control parameters to set, reducing the chances for poor performance. RLS TD is a recursive version of LS TD. LS TD and RLS TD are similar to an algorithm proposed by Werbos, a linear version of Heuristic Dynamic Programming. However, the convergence of linear HDP is more difficult to analyze than it is for LS TD and RLS TD, and it appears experimentally that linear HDP is only locally convergent. That is, linear HDP will converge only if the initial parameter estimates, θ_0 is close enough to the true parameters, θ^*.

We also defined the TD error variance of a Markov chain, σ_{TD}. σ_{TD} is a measure of the noise that is inherent in any TD learning rule, even after the parameters have converged to θ^*. We experimentally conclude that the convergence rate of a TD algorithm depends linearly on σ_{TD}(Figure 4.4). This relationship is very clear for RLS TD, but also seems to hold for NTD(λ) for larger σ_{TD}.

The theorems concerning convergence of LS TD (and RLS TD) can be generalized in at least two ways. First, the transition costs can be random variables instead of constants. $R(x,y)$ would then designate the *expected* cost of a transition from state x to state y. The second change involves the way the states (and state transitions) are sampled. Throughout this chapter we have assumed that the states are visited as the estimation algorithm walks through the Markov chain. This need not be the case. All that is necessary is that there be some limiting distribution π of the states selected for update, such that $\pi_x > 0$ for all states x.

One of the goals of using a parameterized function approximator (of which the linear approximators considered in this chapter are the simplest examples) is to store the value function more compactly than the lookup-table representation. Linear function approximators do not achieve this goal, since they use the same amount of memory as the lookup-table. However, the work presented in this chapter and elsewhere on the performance of TD algorithms with linear function approximators is only the first step toward understanding the performance of TD algorithms with more compact representations.

CHAPTER 5

TEMPORAL DIFFERENCE METHODS FOR LINEAR QUADRATIC SYSTEMS

In this chapter we apply a DP-based reinforcement learning algorithm to the problem of adaptive Linear Quadratic Regulation (LQR). DP-based reinforcement learning algorithms include Sutton's Temporal Difference methods [82], Watkins' Q-learning [92], and Werbos' Heuristic Dynamic Programming [57,94,95,97]. We develop the *Adaptive Policy Iteration* algorithm based on Q-learning and Policy Iteration, and prove that the algorithm converges to the optimal controller provided that the underlying system is controllable and that a particular signal vector is persistently excited. We illustrate the performance of the algorithm by applying it to a model of a flexible beam.

Our convergence proof is one of the first convergence result for DP-based reinforcement learning algorithms applied to problems with continuous state and action spaces. Previous results, as discussed in chapters 2, 3, and 4, are limited to systems with both finite state and finite action sets. Our convergence proof is also one of the first *theoretical* results showing that it is possible to use compact function approximators in conjunction with an IDP algorithm. The only previous convergence proof (of which we are aware) for a DP-based reinforcement learning algorithm applied to a problem with continuous state and action spaces is described by Werbos [96], where he considers using the linear HDP algorithm (see Section 4.6) to evaluate a policy for a linear dynamic system with a *linear* cost function. However, there is no *optimal* policy in this situation, so this result is of limited practical utility. Our Adaptive Policy Iteration algorithm is firmly tied to well–developed LQR theory.

There have been many previous *experimental* demonstrations that compact function approximators may be used in conjunction with IDP algorithms to achieve some level of performance improvement. For example, Tesauro [89, 90] describes a system using TD(λ) that learns to play championship level backgammon (which can be viewed as a Markovian decision task) entirely through self-play. It uses a multilayer perceptron trained using backpropagation as a function approximator. Sofge and White [80] describe a system that learns to improve process control with continuous state and action spaces. Other applications are described by Anderson [1], Jordan and Jacobs [41], and Lin [51,53]. None of these applications, nor many similar applications that have been described, have a firm theoretical grounding. The theory developed in this chapter is a first step toward a firm theoretical grounding for IDP algorithms applied to problems with continuous state and action spaces.

As a demonstration that IDP algorithms developed for finite systems and lookup-table function representations can fail when applied to continuous systems, we describe several versions of Watkins' optimizing Q-learning for the LQR problem. We demonstrate that these algorithms may be unstable, or may converge to controllers that are destabilizing. We then show how to extend the policy iteration algorithm based on Q-learning to the control of systems with hidden state information, and derive a heuristic algorithm for the distributed adaptive control of a class of linear systems.

This chapter extends and discusses more fully the work previously reported by Bradtke [9] and Bradtke, Ydstie, and Barto [10,11].

5.1 Linear Quadratic Regulation

Consider the linear, discrete-time, multivariable, dynamic system [16,56]

$$x_{t+1} = f(x_t, u_t) = Ax_t + Bu_t \tag{5.1}$$

with feedback control

$$u_t = Ux_t. \tag{5.2}$$

Here A, B, and U are matrices of dimensions $n \times n$, $n \times m$, and $m \times n$ respectively and U is chosen so that the matrix $A + BU$ has all of its eigenvalues strictly within the unit circle.

Associated with this system we assign a one step cost:

$$r_t = R(x_t, u_t) = x_t'Ex_t + u_t'Fu_t \tag{5.3}$$

where E is a symmetric positive semidefinite matrix of dimensions $n \times n$ and F is a symmetric positive definite matrix of dimensions $m \times m$. The *total cost* of a state x_t under the control policy U, $V_{\text{U}}(x_t)$, is defined as the discounted sum of all one step costs that will be incurred by using U from time t onward, *i.e.*, $V_{\text{U}}(x_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$, where $0 \leq \gamma \leq 1$ is the discount factor. This definition implies the recurrence relation

$$V_{\text{U}}(x_t) = R(x_t, Ux_t) + \gamma V_{\text{U}}(x_{t+1}). \tag{5.4}$$

V_{U} is a quadratic function [6] and therefore can be expressed as

$$V_{\text{U}}(x_t) = x_t'K_{\text{U}}x_t, \tag{5.5}$$

where K_{U} is the $n \times n$ *cost matrix* for policy U. U^* denotes the policy that is optimal in the sense that the total dicounted cost of every state is minimized. $K^* = K_{\text{U}^*}$ represents the cost matrix associated with U^*.

If the matrices A, B, E, F, and U are known, then K_{U} can be found [6] as the unique positive definite solution to the linear equation

$$K_{\text{U}} = \gamma\,(A + BU)'\,K_{\text{U}}\,(A + BU) + E + U'FU. \tag{5.6}$$

K^* can be found as the unique positive definite solution to the Ricatti equation

$$K^* = A'\left[\gamma K^* - \gamma^2 K^* B\,(F + \gamma B'K^*B)^{-1}\,B'K^*\right] A + E. \tag{5.7}$$

The optimal policy is

$$U^* = -\gamma\,(F + \gamma B'K^*B)^{-1}\,B'K^*A. \tag{5.8}$$

It is a simple but computationally costly matter to derive U^* *if accurate models of the system and cost function are available.* The problem we address is how to define an adaptive policy that converges to U^* *without* access to such models.

5.2 Q-functions and Policy improvement

The recurrence relation for Q-functions given in Equation (2.19) can be rewritten for LQR problems as

$$\begin{aligned} Q_{\mathrm{U}}(x,u) &= R(x,u) + Q_{\mathrm{U}}(f(x,u), U(f(x,u)) \\ &= R(x,u) + Q_{\mathrm{U}}(Ax+Bu, U(Ax+Bu). \end{aligned} \tag{5.9}$$

The Q-function for an LQR problem can be computed explicitly using Equations (5.5) and (2.18). We have

$$\begin{aligned} Q_{\mathrm{U}}(x,u) &= R(x,u) + \gamma V_{\mathrm{U}}(f(x,u)) \\ &= x'Ex + u'Fu + \gamma(Ax+Bu)'K_{\mathrm{U}}(Ax+Bu) \\ &= x'(E+\gamma A'K_{\mathrm{U}}A)x + u'(F+\gamma B'K_{\mathrm{U}}B)u + \gamma x'A'K_{\mathrm{U}}Bu + \gamma u'B'K_{\mathrm{U}}Ax \\ &= \begin{bmatrix} x,u \end{bmatrix}' \begin{bmatrix} E+\gamma A'K_{\mathrm{U}}A & \gamma A'K_{\mathrm{U}}B \\ \gamma B'K_{\mathrm{U}}A & F+\gamma B'K_{\mathrm{U}}B \end{bmatrix} \begin{bmatrix} x,u \end{bmatrix} \\ &= \begin{bmatrix} x,u \end{bmatrix} \begin{bmatrix} H_{\mathrm{U}(11)} & H_{\mathrm{U}(12)} \\ H_{\mathrm{U}(21)} & H_{\mathrm{U}(22)} \end{bmatrix} \begin{bmatrix} x,u \end{bmatrix}' && (5.10) \\ &= \begin{bmatrix} x,u \end{bmatrix}' H_{\mathrm{U}} \begin{bmatrix} x,u \end{bmatrix}, && (5.11) \end{aligned}$$

where $[x,u]$ is the column vector concatenation of x and u and H_{U} is a symmetric positive definite matrix of dimensions $(n+m)\times(n+m)$. The submatrix $H_{\mathrm{U}(22)}$ is symmetric positive definite.

Given the policy U_k and the value function V_{U}, we can we find an improved policy, U_{k+1}, by following Howard [33] in defining U_{k+1} as

$$U_{k+1}x = \operatorname*{argmin}_u \left[r(x,u) + \gamma V_{\mathrm{U}}(f(x,u)) \right],$$

but Equation (2.18) tells us that this can be rewritten as

$$U_{k+1}x = \operatorname*{argmin}_u Q_{\mathrm{U}}(x,u).$$

We can find the minimizing u by taking the partial derivative of $Q_{\mathrm{U}}(x,u)$ with respect to u, setting that to zero, and solving for u. Taking the derivative we get

$$\frac{\partial Q_{\mathrm{U}}(x,u)}{\partial u} = 2(F+\gamma B'K_{\mathrm{U}}B)u + 2\gamma B'K_{\mathrm{U}}Ax.$$

Setting that to zero and solving for u yields

$$u = \underbrace{-\gamma\left(F+\gamma B'K_{\mathrm{U}_k}B\right)^{-1} B'K_{\mathrm{U}_k}A}_{U_{k+1}}\, x.$$

Since the new policy U_{k+1} does not depend on x, it is the minimizing policy for all x. Using Equation (5.10), U_{k+1} can be written as

$$U_{k+1} = -H^{-1}_{\mathrm{U}_k(22)} H_{\mathrm{U}_k(21)}. \tag{5.12}$$

The feedback policy U_{k+1} is by definition a stabilizing policy – it has no higher cost than U_k. A new Q-function can then be assigned to this policy and the policy improvement procedure can be repeated *ad infinitum*.

Earlier work by Kleinman [46] and Bertsekas [6] showed that policy iteration converges for LQR problems. However, the algorithms described Kleinman and Bertsekas required exact knowledge of the system model (Equation (5.1)) and the one-step cost function (Equation (5.3)). The analysis presented here shows how policy iteration can be performed *without* that knowledge. Knowledge of the sequence of functions Q_{U_k} is sufficient.

5.3 Direct Estimation of Q-functions

We now show how the function Q_{U} can be directly estimated using recursive least squares (RLS). It is not necessary to identify either the system model or the one-step cost function separately. First, define the "overbar" function for vectors so that $\overline{x}$ is the vector whose elements are all of the quadratic basis functions over the elements of x, *i.e.*,

$$\overline{x} = \left[x_1^2, \ldots, x_1 x_n, x_2^2, \ldots, x_2 x_n, \ldots, x_{n-1}^2, x_{n-1}x_n, x_n^2\right]'.$$

Next, define the function Θ for square matrices. $\Theta(K)$ is the vector whose elements are the n diagonal entries of K and the $n(n+1)/2 - n$ distinct sums $(K_{ij} + K_{ji})$. The elements of $\overline{x}$ and $\Theta(K)$ are ordered so that $x'Kx = \overline{x}'\Theta(K)$. The original matrix K can be retrieved from $\Theta(K)$ if K is symmetric. If K is not symmetric, then we retrieve the symmetric matrix $\frac{1}{2}(K+K')$, which defines the same quadratic function as K. We can now write

$$Q_{\mathrm{U}}(x,u) = \left[\ x, u\ \right]' H_{\mathrm{U}} \left[\ x, u\ \right] = \overline{\left[\ x, u\ \right]}' \Theta(H_{\mathrm{U}}).$$

Finally, we rearrange Equation (5.9) to yield

$$\begin{aligned}
r_t &= R(x_t, u_t) \\
&= Q_{\mathrm{U}}(x_t, u_t) - \gamma Q_{\mathrm{U}}(x_{t+1}, Ux_{t+1}) \\
&= \left[\ x_t, u_t\ \right]' H_{\mathrm{U}} \left[\ x_t, u_t\ \right] - \gamma \left[\ x_{t+1}, Ux_{t+1}\ \right]' H_{\mathrm{U}} \left[\ x_{t+1}, Ux_{t+1}\ \right] \\
&= \overline{\left[\ x_t, u_t\ \right]}' \Theta(H_{\mathrm{U}}) - \gamma \overline{[x_{t+1}, Ux_{t+1}]}' \Theta(H_{\mathrm{U}}) \\
&= \phi_t' \theta_{\mathrm{U}},
\end{aligned}$$

where

$$\phi_t = \left\{\overline{[x_t, u_t]} - \gamma \overline{[x_{t+1}, Ux_{t+1}]}\right\}, \tag{5.13}$$

and $\theta_{\mathrm{U}} = \Theta(H_{\mathrm{U}})$.

Recursive Least Squares (RLS) can now be used to estimate θ_{U}. The recurrence relations for RLS are given by

$$\hat{\theta}_k(i) = \hat{\theta}_k(i-1) + \frac{P_k(i-1)\phi_t(r_t - \phi_t'\hat{\theta}_k(i-1))}{1 + \phi_t' P_k(i-1)\phi_t} \tag{5.14a}$$

$$P_k(i) = P_k(i-1) - \frac{P_k(i-1)\phi_t\phi_t' P_k(i-1)}{1 + \phi_t' P_k(i-1)\phi_t} \tag{5.14b}$$

$$P_k(0) = P_0. \tag{5.14c}$$

$P_0 = \beta I$ for some large positive constant β. $\theta^*_k = \Theta(H_{\mathrm{U}_k})$ is the true parameter vector for the function Q_{U_k}. $\hat{\theta}_k(i)$ is the i^{th} estimate of θ^*_k. The subscript t and the index i are both incremented at each time step. The reason for the distinction between t and i will be made clear in the next section.

Goodwin and Sin [30] show that this algorithm converges asymptotically to the true parameters if θ^*_k is fixed and ϕ_t satisfies the persistent excitation condition

$$\epsilon_0 I \leq \frac{1}{N}\sum_{i=1}^{N} \phi_{t-i}\phi'_{t-i} \leq \bar{\epsilon}_0 I \qquad \text{for all} \qquad t \geq N_0 \text{ and } N \geq N_0 \tag{5.15}$$

where N_0 and $\epsilon_0 \leq \bar{\epsilon}_0$ are positive numbers. However, asymptotic convergence implies that there will always be some non-zero error in the parameter estimates at any finite stopping time.

The RLS TD algorithm described in Chapter 4 may also be used. In this context it would be represented as

$$\hat{\theta}_k(i) = \hat{\theta}_k(i-1) + \frac{P_k(i-1)\overline{[x_t, u_t]}(r_t - \phi'_t\hat{\theta}_k(i-1))}{1 + \phi'_t P_k(i-1)\overline{[x_t, u_t]}} \tag{5.16a}$$

$$P_k(i) = P_k(i-1) - \frac{P_k(i-1)\overline{[x_t, u_t]}\phi'_t P_k(i-1)}{1 + \phi'_t P_k(i-1)\overline{[x_t, u_t]}} \tag{5.16b}$$

$$P_k(0) = P_0. \tag{5.16c}$$

Here, $\overline{[x_t, u_t]}$ serves as the instrumental variable. The vector $\overline{[x_t, u_t]}$ is correlated with ϕ_t, and since the state transitions are deterministic, it is uncorrelated with the noise in the state transitions. Thus, RLS TD will also converge when applied to deterministic LQR problems.

5.4 Adaptive Policy Iteration for LQR

The policy improvement process based on Q-functions and the ability to estimate H_{U} directly (Section 5.3) are the two key elements of the adaptive policy iteration algorithm. Figure 5.1 gives an outline of the algorithm.

Each policy iteration step consists of two phases: estimation of the Q-function for the current controller, and policy improvement based on that estimate. Consider the k^{th} policy iteration step. U_k is the current controller. The true parameter vector for the function Q_{U_k} is $\theta^*_k = \Theta(H_{\mathrm{U}_k})$. The estimate of θ^*_k at the end of the parameter estimation interval is $\hat{\theta}_k = \hat{\theta}_k(N)$. Each estimation interval is N time-steps long. The RLS algorithm is initialized at the start of the k^{th} estimation interval by setting $P_k(0) = P_0$ and initializing the parameter estimates for the k^{th} estimation interval to the final parameter estimates from the previous interval, *i.e.*, $\hat{\theta}_k(0) = \hat{\theta}_{k-1}(N)$. The index i used in Equation (5.14) counts the number of time steps since the beginning of the estimation interval. After identifying the parameters $\Theta(H_{\mathrm{U}_k})$ for N timesteps,

```
1   Initialize parameters $\hat{\theta}_1(0)$.
2   $t = 0$, $k = 1$.
3   **repeat** forever {
4       Initialize $P_k(0) = P_0$.
5       **for** $i = 1$ to N {
6           $u_t = U_k x_t + e_t$, where $e_t$ is the "exploration" component of the control
            signal.
7           Apply $u_t$ to the system, resulting in state $x_{t+1}$.
8           Update the estimates of the $Q$-function parameters, $\hat{\theta}_k(i)$, using RLS
            (Equation (5.14)).
9           $t = t + 1$.
        }
10      Find the symmetric matrix $\hat{H}_k$ that corresponds to the parameter vector $\hat{\theta}_k$.
11      Perform policy improvement based on $\hat{H}_k$: $U_{k+1} = -\hat{H}^{-1}_{k(22)} \hat{H}_{k(21)}$.
12      Initialize parameters $\hat{\theta}_{k+1}(0) = \hat{\theta}_k$.
13      $k = k + 1$
14  }
```

Figure 5.1 The Q-function based policy iteration algorithm. It starts with the system in some initial state x_0 and with some stabilizing controller U_0. k keeps track of the number of policy iteration steps. t keeps track of the total number of time steps. i counts the number of time steps since the last change of policy. When $i = N$, one policy improvement step is executed.

one policy improvement step is taken based on the estimate $\hat{\theta}_k$. This produces the new controller U_{k+1}, and a new policy iteration step is begun.

Since the k^{th} policy improvement step is based on an *estimate* of $\Theta(H_{U_k})$, it is not clear *a priori* that the sequence U_k will converge to the optimal policy U^*, or even that each of the U_k's is guaranteed to be stabilizing. The convergence proofs of Kleinman [46] and Bertsekas [6] require exact knowledge of the system and take no account of estimation error. Theorem 5.1 establishes that the adaptive policy iteration algorithm presented above does indeed converge, under the appropriate conditions, to the optimal controller. Puterman and Shin consider a similar situation involving value function estimation errors when performing policy iteration for discrete MDPs [65].

Theorem 5.1 (Convergence of adaptive policy iteration) *Suppose that $\{A, B\}$ is a controllable pair, that U_0 is a stabilizing control, and that the vector $\phi(t)$ is persistently excited according to inequality (5.15). Then there exists an estimation interval $N < \infty$ so that the adaptive policy iteration mechanism described above generates a sequence $\{U_k, k = 1, 2, 3, ...\}$ of stabilizing controls, converging so that*

$$\lim_{k \to \infty} \|U_k - U^*\| = 0,$$

where U^* is the optimal feedback control matrix.

Proof: In order to prove this we need a few intermediate results concerning the policy iteration scheme and RLS estimation. The results are summarized below and the proofs are given in Appendix D. First, define the function

$$\sigma(U_k) = \text{trace}(K_{U_k}). \tag{5.17}$$

Lemma 5.1 *If $\{A, B\}$ is controllable, U_1 stabilizing with associated cost matrix K_1 and U_2 is the result of one policy improvement step from U_1, i.e. $U_2 = -\gamma(F + \gamma B'K_1B)^{-1}B'K_1A$, then*

$$\Delta\|U_1 - U_2\|^2 \leq \sigma(U_1) - \sigma(U_2) \leq \delta\|U_1 - U_2\|^2,$$

where

$$0 < \Delta = \underline{\sigma}(F) \leq \delta = \text{trace}\,(F + \gamma B'K_1B)\|\textstyle\sum_{i=0}^{\infty}\gamma^{(i/2)}(A + BU_2)^i\|^2,$$

and $\underline{\sigma}(\cdot)$ denotes the minimum singular value of a matrix.

Lemma 5.2 *If ϕ_t is persistently excited as given by inequaliy (5.15) and $N \geq N_0$, then we have*

$$\|\theta_k^* - \hat{\theta}_k\| \leq \epsilon_N(\|\theta_k^* - \theta_{k-1}^*\| + \|\theta_{k-1}^* - \hat{\theta}_{k-1}\|), \qquad \textit{where } \epsilon_N = \frac{1}{\epsilon_0 N p_0}$$

and p_0 is the minimum singular value of P_0.

Define a scalar "Lyapunov" function candidate

$$s_k = \sigma(U_{k-1}) + \|\theta_{k-2}^* - \hat{\theta}_{k-2}\| \tag{5.18}$$

and suppose that

$$s_i \leq \bar{s}_0 < \infty \qquad \text{for all } 0 \leq i \leq k \tag{5.19}$$

for some upper bound $\bar{s}_0$. From this it follows that U_{k-1} is stabilizing in the sense that

$$\sigma(U_{k-1}) \leq \bar{s}_0 \tag{5.20}$$

and that the parameter estimation error is bounded so that

$$\|\theta_{k-2}^* - \hat{\theta}_{k-2}\| \leq \bar{s}_0. \tag{5.21}$$

It also follows that that the control resulting from a policy update using accurate parameters, U_k^*, is stabilizing and that $\sigma(U^*{}_k) \leq \bar{s}_0$. From continuity of the optimal policy update it then follows that for every $\delta > 0$ there exists $\epsilon_\delta > 0$ so that

$$|\sigma(U) - \sigma(U_k^*)| \leq \delta\|U_k^* - U\| \qquad \text{for all } \|U_k^* - U\| \leq \epsilon_\delta. \tag{5.22}$$

This implies that control laws in a sufficiently small neighborhood around the optimal are stabilizing as well.

We will show that $s_{k+1} \leq s_k$ provided that the estimation interval N, is chosen to be long enough.

Define

$$v_k = \|\theta^*_{k-1} - \hat{\theta}_{k-1}\|,$$

and we get from Lemma 5.2 that for all k

$$v_k \leq \epsilon_N(v_{k-1} + \|\theta^* k - 1 - \theta^* k - 2\|), \tag{5.23}$$

where $\lim_{N\to\infty} \epsilon_N = 0$. Now from the inductive hypothesis (assumption (5.19)) we have

$$v_{k-1} \leq \bar{s}_0 \qquad \text{and} \qquad \|\theta^* k - 2 - \theta^* k - 3\| \leq \kappa_1, \tag{5.24}$$

where κ_1 is a constant. By application of Equation (5.23) we then get

$$v_k \leq \epsilon_N(\bar{s}_0 + \kappa_1). \tag{5.25}$$

It follows that $v_k = \|\theta^* k - 1 - \hat{\theta}_{k-1}\|$ can be made arbitrarily small be choosing the estimation interval N long enough.

$U^*{}_k$ is defined to be the result from applying one step of policy iteration using accurate parameter values, *i.e.*

$$U^*{}_k = -H^{-1}_{\text{k-1(22)}} H_{\text{k-1(21)}}, \tag{5.26}$$

whereas U_k is the feedback law which results from applying the estimated parameters, *i.e.*

$$U_k = -\hat{H}^{-1}_{\text{k-1(22)}} \hat{H}_{\text{k-1(21)}}. \tag{5.27}$$

The matrix inverse is guaranteed to exist when the estimation interval is long enough. From Equations (5.26) and (5.27) we now have

$$U_k - U^*{}_k = -\hat{H}^{-1}_{\text{k-1(22)}} \hat{H}_{\text{k-1(21)}} + H^{-1}_{\text{k-1(22)}} H_{\text{k-1(21)}}.$$

Hence

$$\begin{aligned} U_k - U^*{}_k &= H^{-1}_{\text{k-1(22)}}(H_{\text{k-1(21)}} - \hat{H}_{\text{k-1(21)}}) + (H^{-1}_{\text{k-1(22)}} - \hat{H}^{-1}_{\text{k-1(21)}})\hat{H}_{\text{k-1(21)}} \\ &= H^{-1}_{\text{k-1(22)}}((H_{\text{k-1(21)}} - \hat{H}_{\text{k-1(21)}}) + (\hat{H}_{\text{k-1(22)}} - H_{\text{k-1(22)}})\hat{H}^{-1}_{\text{k-1(22)}} \hat{H}_{\text{k-1(21)}}). \end{aligned}$$

From the definition of θ we have

$$\|\hat{H}_{\text{k-1(22)}} - H_{\text{k-1(22)}}\| \leq \|\theta^* k - 1 - \hat{\theta}_{k-1}\| \qquad \text{and} \qquad \|\hat{H}_{\text{k-1(22)}}\| \leq \|\hat{\theta}_{k-1}\|.$$

It follows that we have

$$\|U_k - U^*{}_k\| \leq \bar{\kappa}_0(1 + \|\hat{\theta}_{k-1}\|) \cdot \|\theta^* k - 1 - \hat{\theta}_{k-1}\|,$$

where $\bar{\kappa}_0$ is a positive constant, provided that N is sufficiently large. Since the estimated parameters are bounded it follows that there exists another constant κ_0 so that

$$\|U_k - U^*{}_k\| \leq \kappa_0\|\theta^* k - 1 - \hat{\theta}_{k-1}\| = \kappa_0 v_k. \tag{5.28}$$

It follows from Equation (5.25) that we have

$$\|U_k - U^*{}_k\| \le \epsilon_N \kappa_0 (\bar{s}_0 + \kappa_1). \tag{5.29}$$

It then follows from Equation (5.22) that

$$|\sigma(U_k) - \sigma(U^*{}_k)| \le \delta \|U_k^* - U_k\| \qquad \text{for all } N \text{ such that } \epsilon_N \kappa_0 (\bar{s}_0 + \kappa_1) \le \epsilon_\delta.$$

This implies that U_k is stabilizing if N is large enough and that there exists an integer N_1 and an associated constant $\bar{\delta}$, so that

$$|\sigma(U_k) - \sigma(U_{k-1})| \le \bar{\delta} \|U_k - U_{k-1}\| \qquad \text{for all } N \ge N_1.$$

In other words, if the estimation interval is long enough, then the difference between two consecutive costs is bounded by the difference between two consecutive controls. We use the definition of the parameter estimation vector to write this as

$$\|\theta^* k - 1 - \theta^* k - 2\| \le \delta_1 \|U_k - U_{k-1}\|^2 \qquad \text{for all } N \ge N_1, \tag{5.30}$$

where δ_1 is a constant. We now re-write Equation (5.30) as

$$\|\theta^* k - 1 - \theta^* k - 2\| \le 2\delta_1 (\|U_k^* - U_{k-1}\|^2 + \|U_k^* - U_k\|^2).$$

From inequality (5.28) and the definition of v_k, we then get

$$\|\theta^* k - 1 - \theta^* k - 2\| \le 2\delta_1 (w_k^2 + \kappa_0 v_k), \tag{5.31}$$

where

$$w_k = \|U_k^* - U_{k-1}\|.$$

By combining Equations (5.23) and (5.31) we then get

$$v_k \le \epsilon_N (v_{k-1} + 2\delta_1 (w_k^2 + \kappa_0 v_k)),$$

which we re-write as

$$v_k \le \epsilon_N \mu_N (v_{k-1} + 2\delta_1 w_k^2), \tag{5.32}$$

where

$$\mu_N = (1 - 2\delta_1 \kappa_0 \epsilon_N)^{-1}.$$

According to the assumption we can choose N large enough so that $0 < \mu_N < \infty$. This gives a recursion for v_k. The critical point to notice is that v_k has a strong stability property when the estimation interval is long. The parameter $\epsilon_N \mu$ is then small since ϵ_N converges uniformly to 0 and μ towards 1.

We now develop the recursion for $\sigma(U_k)$. First we have

$$\sigma(U_k) - \sigma(U_{k-1}) = \sigma(U^*{}_k) - \sigma(U_{k-1}) + \sigma(U_k) - \sigma(U^*{}_k). \tag{5.33}$$

From Equation (5.33) and Lemma 5.1, using Equation (5.22) again, it follows that we can choose the update interval so that we have a constant δ_2 so that

$$\sigma(U_k) - \sigma(U_{k-1}) \le -\Delta \|U_k^* - U_{k-1}\|^2 + \delta_2 \|U_k^* - U_k\|^2.$$

Using Equation (5.28) we then get

$$\sigma(U_k) - \sigma(U_{k-1}) \le -\Delta \|U_k^* - U_{k-1}\|^2 + \delta_2 \kappa_0 \|\theta^* k - 1 - \hat{\theta}_{k-1}\|$$

$$\leq -\Delta w_k^2 + \delta_2 \kappa_0 v_k.$$

By using Equation (5.32) and the recursion for v_k we then have

$$\sigma(U_k) - \sigma(U_{k-1}) \leq -\Delta w_k^2 + \delta_1 \kappa_0 \epsilon_N \mu_N (v_{k-1} + 2\delta_2 w_k^2). \tag{5.34}$$

Equation (5.32) and Equation (5.34) together define the system

$$\begin{bmatrix} v_k \\ \sigma(U_k) \end{bmatrix} = \begin{bmatrix} \epsilon_N \mu_N & 0 \\ \delta_2 \kappa_0 \epsilon_N \mu_N & 1 \end{bmatrix} \begin{bmatrix} v_{k-1} \\ \sigma(U_{k-1}) \end{bmatrix} + \begin{bmatrix} 2\epsilon_N \mu_N \delta_2 \\ -\Delta + 2\delta_2 \kappa_0 \epsilon_N \mu_N \end{bmatrix} w_k^2.$$

In order to study this system we defined the function

$$s_k = \sigma(U_{k-1}) + v_{k-1}.$$

From the above we then have

$$s_{k+1} = s_k + (-1 + \epsilon_N \mu_N (1 + \delta_2 \kappa_0)) v_{k-1} + (-\Delta + 2\epsilon_N \mu_N \delta_2 (1 + \kappa_0)) w_k^2.$$

It now suffices to choose N so that ϵ is small enough to give

$$\begin{aligned} 1 - \epsilon_N \mu_N (1 + \delta_2 \kappa_0) &= \epsilon_1 > 0 \\ \Delta - 2\epsilon_N \mu_N \delta_2 (1 + \kappa_0) &= \epsilon_2 > 0. \end{aligned}$$

We then get

$$s_{k+1} = s_k - \epsilon_1 v_{k-1} - \epsilon_2 w_k^2 \leq s_k.$$

From this we conclude that $s_{k+1} \leq s_k$ and using induction we finally have

$$\epsilon_1 \sum_{k=1}^{\infty} v_k \leq \bar{s}_0 \qquad \text{and } \epsilon_2 \sum_{k=1}^{\infty} w_k^2 \leq \bar{s}_0.$$

The result now follows since U_0 is stabilizing.
Q.E.D.

5.5 Demonstration of the adaptive policy iteration algorithm in simulation

Figure 5.2 illustrates the flexible beam used to demonstrate the performance of the adaptive policy iteration algorithm. The beam system is a 20-dimensional discrete-time approximation of an Euler-Bernoulli flexible beam supported at both ends. The state of the system consists of the displacement from the horizontal of each of the $n = 10$ nodes, and the velocity of each of the nodes. There is one control point. The scalar control signal is the acceleration applied at that point. The regulation task facing the controller is to drive the system state to zero, *i.e.*, to damp out all vibration from the system and to leave the beam perfectly flat.

Figures 5.3, 5.4, and 5.5 demonstrate the performance of the algorithm. The initial policy U_0 is an arbitrarily selected stabilizing controller for the system. The

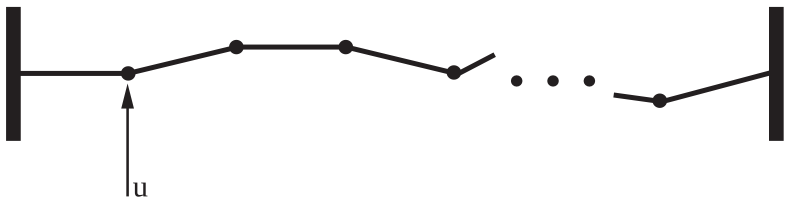


Figure 5.2 Illustration of the flexible beam. The beam system is a 20-dimensional discrete-time approximation of an Euler-Bernoulli flexible beam supported at both ends. The state of the system consists of the displacement from the horizontal and the velocity of each of the $n = 10$ nodes. There is one control point. The scalar control signal is the acceleration applied at that point.

starting point x_0 is a random point in a neighborhood around $0 \in \mathbb{R}^{20}$. We used a random exploration signal generated from a normal distribution in order to induce persistent excitation of the vector ϕ_t. Although this method of inducing persistent excitation does not match the requirements of Theorem 5.1, it is very simple to implement, and has worked very well in practice. There are 231 parameters to be estimated for this system, so we set $N = 500$, approximately twice that. Figure 5.3 shows the max norm of the difference between the current controller and the optimal controller at each policy iteration step. Figure 5.4 shows the max norm of the difference between the estimate of the Q-function parameters for the current controller and the Q-function parameters for the optimal controller at each policy iteration step. After only eight policy iteration steps the adaptive policy iteration algorithm has converged close enough to U^* and H^* that further improvements are limited by the machine precision. Although this demonstration is for a single-input system, the algorithm performs equally well on multi-input systems. Figure 5.5 shows the max norm of the state vector x_t as a function of time. A new controller is installed every 500 time steps. Although each of the controllers in the sequence is stabilizing, and the series converges to the optimal controller, the state is not driven to 0. This is due to the necessity of maintaining persistent excitation. In practice, adaptation could cease after some finite number of policy iteration steps, when the sequence of controllers is judged to have converged.

Figures 5.6 and 5.7 show the results of experiments demonstrating that the adaptive policy iteration algorithm can fail when the assumptions of Theorem 5.1 are violated. That is, when either persistent excitation is not maintained, or when the estimation interval, N, is too short. The demonstration system is the same discretized beam used above. Figure 5.6 shows the results of violating the persistent excitation assumption. As in the experiment described above, policy improvement steps were performed every 500 time steps. However, the exploratory signal was a constant zero, so ϕ_t was not persistently excited. The graph shows the size of $\|x_t\|_\infty$ growing rapidly to infinity after the first policy "improvement" step at time 500. The lack of persistent excitation prevented $\hat{H}_{U_1}$ from being an adequate approximation to H_{U_1}, causing the "improved" controller, U_2, to be destabilizing. Figure 5.7 shows the results of a too short estimation interval. In this experiment, policy improvement was performed every $N = 100$ time-steps instead of every 500 time-steps. Since

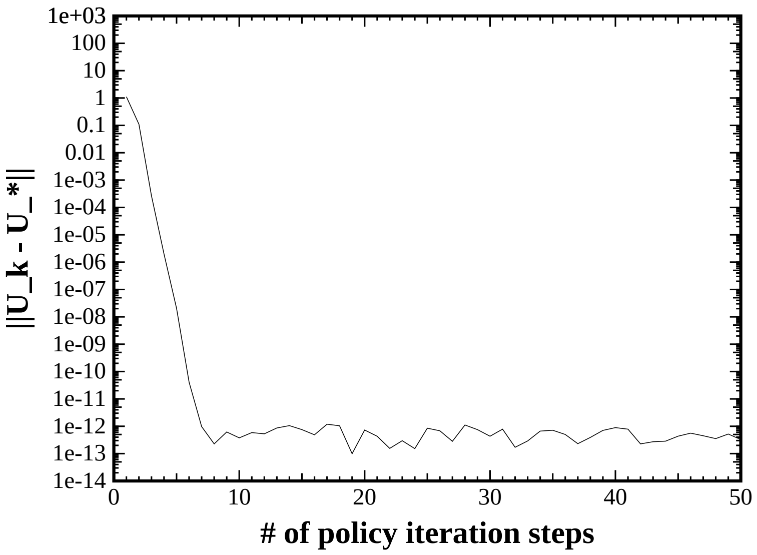


Figure 5.3 Performance of the adaptive policy iteration algorithm: policy convergence. Convergence was measured using the norm of the difference between the optimal policy and the current estimate of the optimal: $\|U_k - U^*\|_\infty$. The demonstration system is a 20-dimensional discrete-time approximation of an Euler-Bernoulli flexible beam supported at both ends. There is one control point. U_0 is an arbitrarily selected stabilizing controller for the system. x_0 is a random point in a neighborhood around $0 \in \mathbb{R}^{20}$.

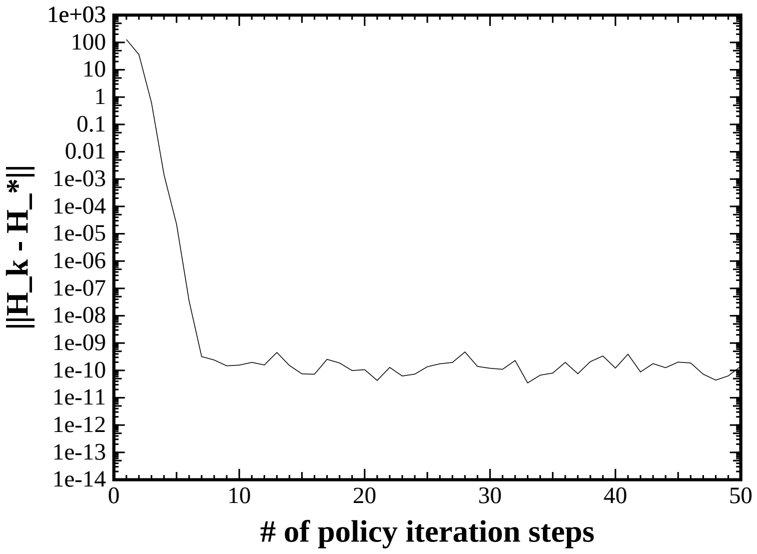


Figure 5.4 Performance of the adaptive policy iteration algorithm: parameter convergence. Convergence was measured using the norm of the difference between the parameters of the optimal policy and the current estimate of the optimal parameters: $\|\hat{H}_{U_k} - H^*\|_\infty$. The demonstration system is a 20-dimensional discrete-time approximation of an Euler-Bernoulli flexible beam supported at both ends. There is one control point. U_0 is an arbitrarily selected stabilizing controller for the system. x_0 is a random point in a neighborhood around $0 \in \mathbb{R}^{20}$.

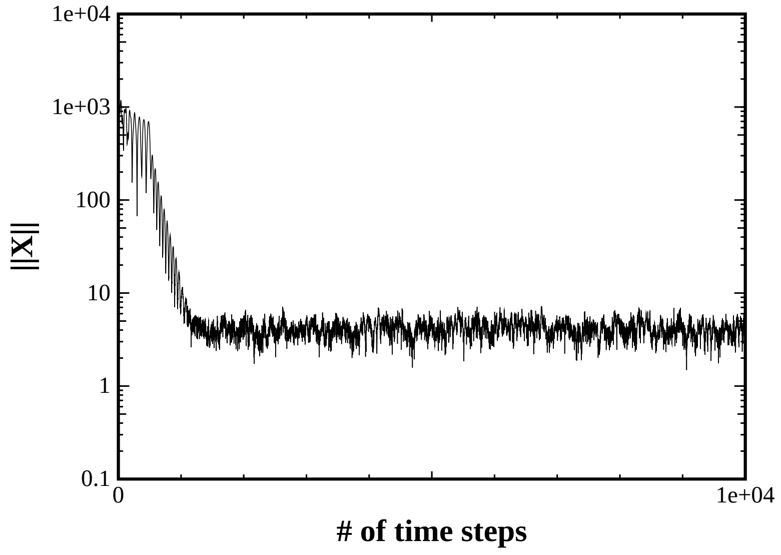


Figure 5.5 Performance of the adaptive policy iteration algorithm: regulation performance. The demonstration system is a 20-dimensional discrete-time approximation of an Euler-Bernoulli flexible beam supported at both ends. There is one control point. U_0 is an arbitrarily selected stabilizing controller for the system. x_0 is a random point in a neighborhood around $0 \in \mathbb{R}^{20}$. The goal of regulation is to drive the system state to 0, but the necessity for persistent excitation keeps the state away from the goal.

there are 231 parameters to be estimated the estimator could not have formed a good approximation to all of them. The graph shows that the controller that resulted from the first policy "improvement" step was destabilizing in this situation also.

5.6 Comparison with model-based adaptive optimal control

The standard method of adaptively finding the LQ optimal controller for an unknown system is based on building a model of the system and then using that model under the certainty equivalence assumption to determine the optimal controller [30]. Figure 5.8 gives a version of the algorithm that builds a state-space model as described in Section 5.1. The A and B matrices of Equation (5.1) contain a total of $n^2 + nm$ independent parameters that must be estimated. On the other hand, the function Q_U for some policy U contains $(n+m)(n+m+1)/2$ independent parameters. The adaptive policy iteration algorithm estimates fewer parameters than the model-based method whenever

$$\frac{(n+m)(n+m+1)}{2} < n^2 + nm.$$

Solving, we see that the adaptive policy iteration algorithm requires estimating fewer parameters whenever $m < n - 1$. Otherwise, the model-based method requires fewer parameters.

This analysis does not take into account the substantial computational costs undertaken by the model-based method to solve the Ricatti equation (Equation (5.7)) at every time step in order to find the certainty equivalent optimal controller $\hat{U}_t$.

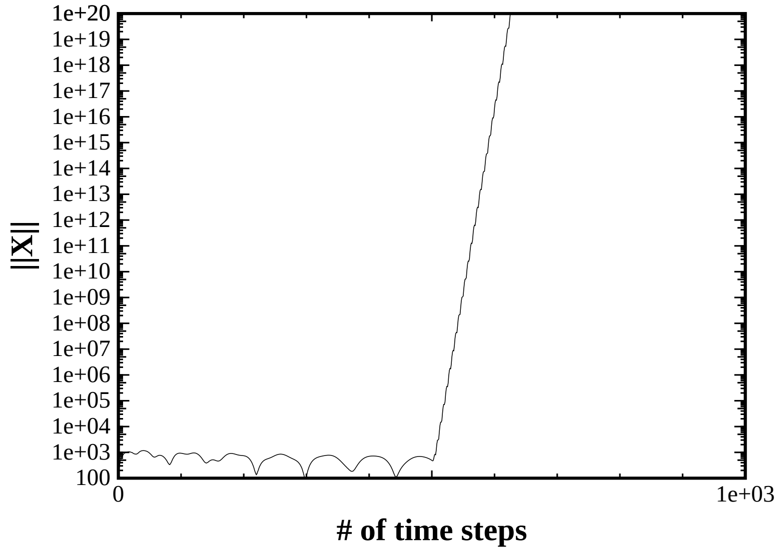


Figure 5.6 The impact of insufficient excitation on the performance of the adaptive policy iteration algorithm. The graph shows the size of $\|x_t\|_\infty$ growing rapidly to infinity after the first policy "improvement" step at time 500. The lack of persistent excitation prevented $\hat{H}_{U_1}$ from being an adequate approximation to H_{U_1}, causing the "improved" controller, U_2, to be destabilizing.

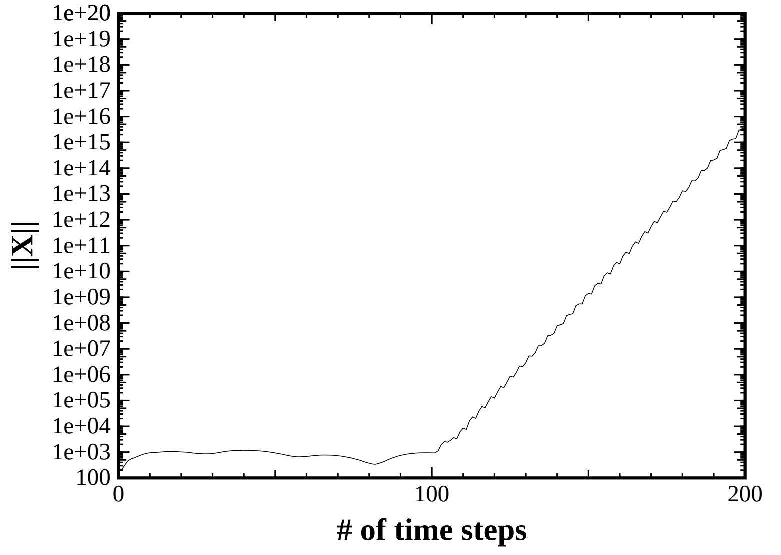


Figure 5.7 The impact of a short estimation interval on the performance of the adaptive policy iteration algorithm. In this experiment, policy improvement was performed every $N = 100$ time-steps instead of every 500 time-steps. Since there are 231 parameters to be estimated, this interval is too short. The graph shows the size of $\|x_t\|_\infty$ growing rapidly to infinity after the first policy "improvement" step at time 100. The short estimation interval prevented $\hat{H}_{U_1}$ from being an adequate approximation to H_{U_1}, causing the "improved" controller, U_2, to be destabilizing.

The adaptive policy iteration algorithm has no such burden, so that it may have a computational advantage over the model-based method even when $m \geq n - 1$.

```
1  Initialize the estimates $\hat{A}_0$ of $A$, $\hat{B}_0$ of $B$, and $\hat{U}_0$ of $U^*$.
2  for $t = 0$ to $\infty$ {
3      Apply the control signal specified by $\hat{U}_t$ to the system, resulting in a change
       of state from $x_t$ to $x_{t+1}$.
4      Use the experienced state transition to update $\hat{A}_t$ to $\hat{A}_{t+1}$ and $\hat{B}_t$ to $\hat{B}_{t+1}$.
5      Use $\hat{A}_{t+1}$ and $\hat{B}_{t+1}$ to determine $\hat{U}_{t+1}$ as described in Equation (5.7) and
       Equation (5.8).
6  }
```

Figure 5.8 State-space model based adaptive optimal control. As $\hat{A}_t$ approaches A and $\hat{B}_t$ approaches B, $\hat{U}_t$ approaches U^*.

5.7 The Failure of Optimizing Q-learning for LQR

Policy iteration would seem to be an unnecessarily slow and complicated method of finding the optimal controller, U^*. Every controller U_k in the sequence must be evaluated before the next can be derived. Instead, why not do as Watkins' optimizing Q-learning rule does (Equation (2.22)), and try to learn Q^* (and thereby U^*) directly? Figures 5.9 and 5.10 give two versions of an optimizing Q-learning algorithm designed to estimate the optimimal Q-function for an LQR problem directly. Like the adaptive policy iteration algorithm discussed above, these algorithms are designed to be used on-line, under the assumption that the system model is unavailable. The only difference between the two versions of the optimizing Q-learning algorithm lies in the choice of controller that is used during training. The first version, designated as Algorithm A in Figure 5.9, always follows some prespecified stabilizing controller U_0. The second version, designated as Algorithm B in Figure 5.10, follows the certainty equivalent optimal policy at every time step. Under the assumption that the parameters $\hat{H}_{t-1}$ are a good estimate of H^*, the certainty equivalent optimal policy at step t is the policy defined by

$$U_t = -\hat{H}^{-1}_{t\text{-}1(22)} \hat{H}_{t\text{-}1(21)}.$$

What are the convergence properties of these algorithms? If Algorithm A converges to some fixed point, that fixed point must satisfy the equation

$$[x,u]' \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} [x,u] =$$

```
1  Initialize parameters $\hat{\theta}(0)$.
2  Initialize $P(0) = P_0$.
3  for $t = 0$ to $\infty$ {
4      $u_t = U_0 x_t + e_t$, where $e_t$ is the "exploration" component of the control signal.
5      Apply $u_t$ to the system, resulting in state $x_{t+1}$.
6      Define $a_{t+1} = -\hat{H}_{t(22)}^{-1} \hat{H}_{t(21)} x_{t+1}$.
7      Update the $Q$-function parameters, $\hat{H}_t$, using the Recursive Least Squares
       implementation of the optimizing $Q$-learning rule, Equation (2.22).
8  }
```

Figure 5.9 An optimizing Q-learning algorithm for LQR: Algorithm A. Algorithm A starts with the system in some initial state x_0 and with some stabilizing controller U_0. t keeps track of the total number of time steps. Unlike the adaptive policy iteration algorithm shown in Figure 5.1, U_0 is followed throughout training.

$$x'Ex + u'Eu + \gamma[Ax + Bu, a]' \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} [Ax + Bu, a], \qquad (5.35)$$

where $a = -H_{22}^{-1} H_{21}(Ax + Bu)$. Equation (5.35) actually specifies $(n+m)(n+m+1)/2$ polynomial equations in $(n+m)(n+m+1)/2$ unknowns (remember that H is symmetric). We know that there is at least one solution, that corresponding to the optimal policy, but there may be other solutions as well.

As an example of the possibility of multiple solutions, consider the 1-dimensional system with $A = B = E = F = [1]$ and $\gamma = 0.9$. Substituting these values into Equation (5.35) and solving for the unknown parameters yields two solutions. They are

$$\begin{bmatrix} 2.4296 & 1.4296 \\ 1.4296 & 2.4296 \end{bmatrix} \text{ and } \begin{bmatrix} 0.3704 & -0.6296 \\ -0.6296 & 0.3704 \end{bmatrix}.$$

The first solution is H^*. The second solution, if used to define an "improved" policy as describe in Section 5.2, results in a destablizing controller. This is certainly not a desirable result. Experiments show that the algorithm in Figure 5.9 will converge to either of these solutions if the initial parameter estimates, $\hat{\theta}(0)$, are close enough to that solution.

A fixed point of Algorithm B must satisfy the same equation as a fixed point of Algorithm A. However the behavior of the two algorithms will differ markedly should they converge to the solution that corresponds to an unstable controller. Algorithm A always follows the original, stabilizing controller, so the behavior of the controlled system always remains stable. However, Algorithm B always follows the certainty equivalent optimal controller. Therefore, if it should converge to the unstable solution, the controlled system will be destabilized.

```
Initialize parameters $\hat{\theta}(0)$.
Initialize $P(0) = P_0$.
for $t = 0$ to $\infty$ {
    Define $U_t = -\hat{H}^{-1}_{t\text{-}1(22)} \hat{H}_{t\text{-}1(21)}$.
    $u_t = U_t x_t + e_t$, where $e_t$ is the "exploration" component of the control signal.
    Apply $u_t$ to the system, resulting in state $x_{t+1}$.
    Define $a_{t+1} = U_t x_{t+1}$.
    Update the $Q$-function parameters, $\hat{H}_t$, using the Recursive Least Squares
    implementation of the optimizing $Q$-learning rule, Equation (2.22).
}
```

Figure 5.10 An optimizing Q-learning algorithm for LQR: Algorithm B. Algorithm B starts with the system in some initial state x_0 and with some stabilizing controller U_0. t keeps track of the total number of time steps. The algorithm follows the certainty equivalent optimal policy at every time step.

This analysis of Algorithms A and B shows that a naive translation of using Watkins' Q-learning rule directly to the LQR domain will not necessarily converge to the optimal Q-function. Convergence is critically dependent upon the initial parameter estimates, $\hat{\theta}(0)$. More generally, this demonstrates that there are pitfalls to applying the theory of Incremental Dynamic Programming as developed for discrete, finite MDPs and lookup-table function approximators directly to problems where these assumptions are violated.

5.8 Adaptive Optimal Control of the Input-Output Behavior

In the systems we have considered up to this point (described in Equation (5.1)), the state of the system is observable at every time step. A more general system model allows the state to be hidden, only allowing some linear function of the state to be directly observable. Equations (5.36a) and (5.36b) give such a model. This is a deterministic multi-input/multi-output (MIMO) system with observable output vectors y_t.

$$x_{t+1} = Ax_t + Bu_t \tag{5.36a}$$

$$y_t = Cx_t + Du_t \tag{5.36b}$$

$$u_t = Ux_t. \tag{5.36c}$$

y_t is the information available to an observer at time step t. If the matrix C is not of full rank, then y_t alone will not contain enough information to reconstruct the state of the system. We show in this section how to apply the adaptive policy iteration algorithm to deterministic MIMO systems with hidden state.

Let us assume that there is some sequence of controls being applied to the system. Then we can build a difference equation model of the system based only on the observable information: the input and output vectors.

$$A_0 y_t + \sum_{i=1}^{k} A_i y_{t-i} = \sum_{i=1}^{l} B_i u_{t-i}. \tag{5.37}$$

Equation (5.37) is a deterministic version of an ARMAX (autoregressive, moving average, with exogenous inputs) process [55, 104]. Assuming that A_0 is invertible, Equation (5.37) can be rewritten as

$$y_t = -A_0^{-1} \sum_{i=1}^{k} A_i y_{t-i} + A_0^{-1} \sum_{i=1}^{l} B_i u_{t-i},$$

or, by renaming $A_0^{-1} A_i$ to be simply A_i and $A_0^{-1} B_i$ to be B_i,

$$y_t = -\sum_{i=1}^{k} A_i y_{t-i} + \sum_{i=1}^{l} B_i u_{t-i+1}. \tag{5.38}$$

There is a unique minimum size ARMAX model that will model the output behavior of the system described by Equations (5.36), that is, a model where k and l are as small as possible. Non-minimum size models, in which k and/or l are overestimated, can also accurately track the output behavior of the underlying system. However, the non-minimum size models are not unique.

We can transform the ARMAX model in Equation (5.38) into a state space model like Equation (5.1) using the following definitions [2]. Define $\tilde{x}_t$, $\tilde{A}$, and $\tilde{B}$ as

$$\tilde{x}_t = \begin{bmatrix} y_{t-1} \\ \cdots \\ y_{t-k} \\ u_t \\ \cdots \\ u_{t-l+1} \end{bmatrix}$$

$$\tilde{A} = \begin{bmatrix} -A_1 & -A_2 & \cdots & -A_{k-1} & -A_k & B_1 & B_2 & \cdots & B_{l-1} & B_l \\ I & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & I & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & I & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & I & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & I & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & I & 0 \end{bmatrix}$$

$$\tilde{B} = \begin{bmatrix} 0 \\ \cdots \\ 0 \\ I \\ 0 \\ \cdots \\ 0 \end{bmatrix} \begin{array}{l} \left.\vphantom{\begin{matrix}0\\ \cdots\\ 0\end{matrix}}\right\} k \text{ submatrices} \\ \\ \left.\vphantom{\begin{matrix}0\\ \cdots\\ 0\end{matrix}}\right\} l-1 \text{ submatrices} \end{array} .$$

Then we have

$$\tilde{x}_{t+1} = \tilde{A}\tilde{x}_t + \tilde{B}u_t. \tag{5.39}$$

Therefore, assuming that $\tilde{A}$ and $\tilde{B}$ meet the requirements of Theorem 5.1, and that some stabilizing $\tilde{U}_0$ can be found such that

$$u_t = \tilde{U}_0\tilde{x}_t, \tag{5.40}$$

then we can apply the Q-learning policy iteration algorithm to the MIMO system in order to find the optimal $\tilde{U}_*$. Furthermore, as long as the system given by Equation (5.39) remains stable, then the underlying system will remain stable. Of course, now the cost at each time step will be a function of $\tilde{x}_t$ and u_t:

$$r_t = R(\tilde{x}_t, u_t) = \tilde{x}_t' E\tilde{x}_t + u_t' F u_t. \tag{5.41}$$

The only problem with this approach to the adaptive control of a MIMO system is that the delay lengths k and l from equation Equation (5.38) will, in general, be unknown. We would like the algorithm to be robust in the case of overestimating k and/or l. Theorem 5.1 tells us that the algorithm will be robust when overestimating the order of the system as long as the pair $\{\tilde{A}, \tilde{B}\}$ is controllable[1]. Figures 5.11, 5.12, and 5.13 show the results of a set of experiments to verify this empirically. The underlying system is given by

$$A = \begin{bmatrix} 1.0 & 0.1 \\ 0.0 & 1.0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0.005 \\ 0.1 \end{bmatrix}, \qquad C = \begin{bmatrix} 1.0 & 0.0 \end{bmatrix}, \qquad \text{and} \qquad D = 0.0.$$

The E and F matrices were chosen to be identity matrices of the appropriate sizes for each of the experiments. Each of the three experiments starts the same initial controller, $u_t = U_0 x_t$, where $U_0 = \begin{bmatrix} -2.0 & -3.0 \end{bmatrix}$. Although this initial controller has access to the state of the system, the adaptive policy iteration algorithm does not. The improved controllers that are generated by the adaptive policy iteration algorithm are all of the form $u_t = U_i \tilde{x}_t$. The experiment shown in Figure 5.11 sets

[1]The pair $\{A, B\}$ is *controllable* if the $n \times nm$ matrix $\begin{bmatrix} B, AB, A^2B, \ldots, A^{n-1}B \end{bmatrix}$ has linearly independent rows.

$k = l = 2$, the minimum values for this system. The minimum size ARMAX model for this system is

$$y_t = 2.0y_{t-1} - y_{t-2} + .005u_t + .005u_{t-1}. \tag{5.42}$$

The experiment shown in Figure 5.12 sets $k = l = 3$. There are infinitely many ARMAX models of order 3, each of the form

$$\begin{aligned} y_t &= (2.0 - \alpha)y_{t-1} - y_{t-2} + .005u_t + .005u_{t-1} \\ &\quad +\alpha \left[\; y_{t-2} - y_{t-3} + .005u_{t-1} + .005u_{t-2} \; \right] && (5.43) \\ &= (2.0 - \alpha)y_{t-1} + (2\alpha - 1)y_{t-2} - \alpha y_{t-3} \\ &\quad +.005u_t + .005(\alpha + 1)u_{t-1} + .005\alpha u_{t-2}, && (5.44) \end{aligned}$$

where α is some constant, and the quantity within the brackets equals y_{t-1} by application of Equation (5.42). The experiment shown in Figure 5.11 sets $k = l = 5$. There are also infinitely many ARMAX models of order 5, the form of which can be similarly derived from Equation (5.42).

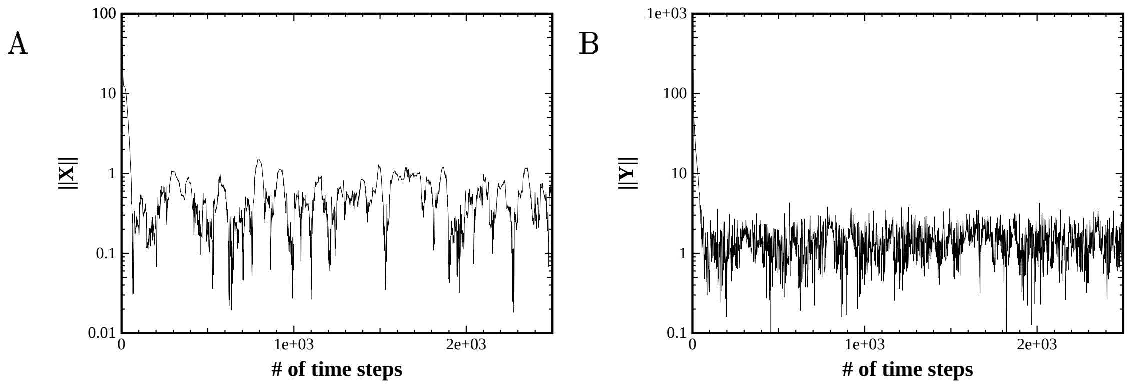


Figure 5.11 SISO adaptive optimal control: delay 2. This is a demonstration of Q-learning based adaptive optimal control on the discretized double integrator with hidden state. $k = l = 2$, the minimum values for this system. Panel A shows the norm of the state, x_t, at each time step. Panel B shows the norm of the delayed coordinate representation, $\tilde{x}_t$, at each time step. Policy iteration was performed every 100 time steps.

Panel A of each figure shows the norm of the state, x_t, at each time step. Panel B shows the norm of the delayed coordinate representation, $\tilde{x}_t$, at each time step. Policy iteration was performed every 100 time steps. It is obvious that the controllers remained stabilizing, even though k and l were overestimated in the second and third experiments. Comparison of panel A in all three experiments shows that the state trajectory was virtually identical in all three cases. We would expect the initial portion of each trajectory to be identical in any case, since all three experiments started with the same initial controller. However, the fact that the trajectories match

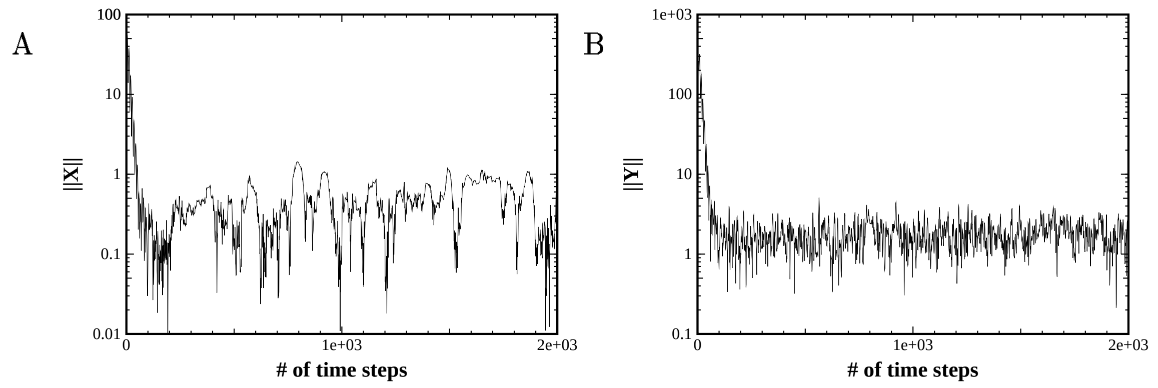


Figure 5.12 SISO adaptive optimal control: delay 3. This is a demonstration of Q-learning based adaptive optimal control on the discretized double integrator with hidden state. $k = l = 3$, higher than the true system order. Panel A shows the norm of the state, x_t, at each time step. Panel B shows the norm of the delayed coordinate representation, $\tilde{x}_t$, at each time step. Policy iteration was performed every 100 time steps. The overestimation does not inhibit the ability to learn an improved controller.

so closely thereafter argues for the conclusion that the controller sequence found by the adaptive policy iteration algorithm for one experiment is nearly the same in effect on the state as the controller sequence found in each of the other two experiments, despite the fact that the delayed coordinate representations are so different for each experiment.

Let us consider the ARMAX model of order 3 more fully. From Equation (5.44) and the definitions of $\tilde{A}$ and $\tilde{B}$, we have

$$\tilde{A}_3 = \begin{bmatrix} (2.0-\alpha) & (2\alpha-1) & -\alpha & .005 & .005(\alpha+1) & .005\alpha \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \text{ and } \tilde{B}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

Testing to see whether the pair $\{\tilde{A}_3, \tilde{B}_3\}$ is controllable, we get that

$$\left[\tilde{B}_3, \tilde{A}_3\tilde{B}_3, \tilde{A}_3^2\tilde{B}_3, \ldots, \tilde{A}_3^5\tilde{B}_3 \right] = \begin{bmatrix} 0 & .005 & .015 & .025 & .035 & .045 \\ 0 & 0 & .005 & .015 & .025 & .035 \\ 0 & 0 & 0 & .005 & .015 & .025 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

which is singular. Therefore, the system $\{\tilde{A}_3, \tilde{B}_3\}$ is not controllable for any value of α. This violates the controllability condition of Theorem 5.1. Despite this, the adaptive

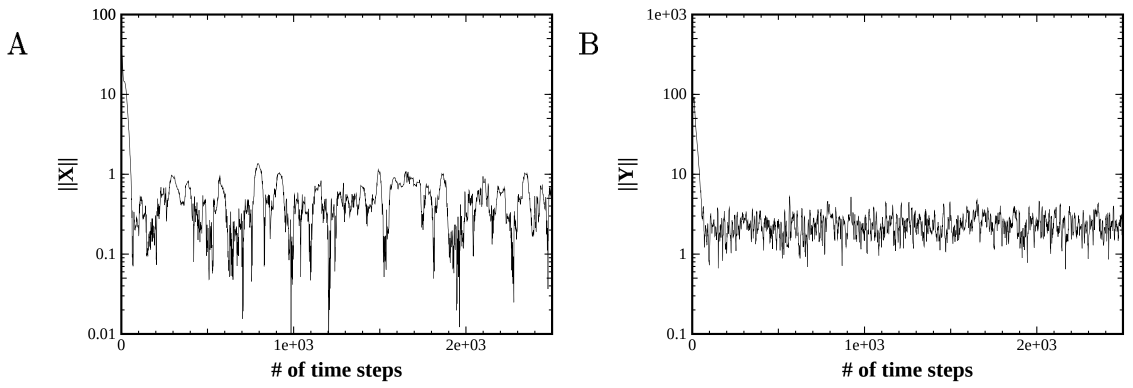


Figure 5.13 SISO adaptive optimal control: delay 5. This is a demonstration of Q-learning based adaptive optimal control on the discretized double integrator with hidden state. $k = l = 5$, higher than the true system order. Panel A shows the norm of the state, x_t, at each time step. Panel B shows the norm of the delayed coordinate representation, $\tilde{x}_t$, at each time step. Policy iteration was performed every 100 time steps. The overestimation does not inhibit the ability to learn an improved controller.

policy iteration algorithm managed to converge to a stabilizing policy. Testing the algorithm with different initial estimates for H_{U_0} caused convergence to different final estimates for H^*. However, in each case the policy implied by these different Q-functions (see Equation (5.12)) was the same, so it seems that there is still a unique convergence point despite the violation of controllability. This has not been verified theoretically.

5.9 Distributed Adaptive Control

Lightweight and flexible structures are becoming increasingly important under the dictates of economics and improved materials technology. Examples include bridges, tall buildings, large space structures, and extended links of robot arms. Many of these structures can be modelled as high-dimensional, multi-input, multi-output linear systems. Depending on the size of the system, the design of a Linear-Quadratic (LQ) optimal controller can be prohibitively expensive. This is true for both traditional off-line techniques as outlined in Section 5.1 and for the on-line reinforcement learning techniques described previously in this chapter. In this section we describe a distributed version of the adaptive policy iteration algorithm, and experimentally verify that it can be used to derive an approximation to the optimal controller under certain circumstances. The savings in computational cost per time step can be substantial.

We make two assumptions in our derivation of the distributed adaptive policy iteration algorithm. The first assumption is that, given a loosely coupled control rule, the closed-loop state transition function of the linear system is loosely coupled. This means that changes in component i of the state vector x from time t to time $t+1$ depend primarily on the values of components near i. The second assumption is that the one-step quadratic cost function is also loosely coupled. This implies that each component of x and u interacts primarily with nearby components in determining the cost incurred at each time step.

Given these assumptions, we partition the state and action spaces, assigning an adaptive Q-learning controller to each partition. Each controller receives only partial state information, and contributes only a portion of the global control signal. Each controller also computes a local reinforcement signal using the relevant portion of the global one-step quadratic cost function. Each controller uses the algorithm described in Section 5.4 to find a controller that optimizes its local costs. Figure 5.14 gives a pseudocode sketch of the distributed algorithm. There is no direct communication between the controllers. The presence of other controllers is seen only through their effect on the locally sensed state information. The interaction between controllers is minimized by the diagonal nature of the system and cost function. The controllers adapt in parallel, deriving "improved" control rules simultaneously. The algorithm is distributed, but not asynchronous.

Flexible structures are good candidates for systems that match our two assumptions. If the control rule U is loosely coupled, then the closed-loop state transition function will also be loosely coupled. The distributed control law implemented by this algorithm is loosely coupled by design. Choosing a loosely coupled quadratic cost function is simple, and does not depend on the system.

The distributed adaptive policy iteration algorithm is heuristic in that there is as yet no theory that gives precise conditions under which it is guaranteed to remain stable or to converge to a best approximation to the optimum centralized control rule. Although this algorithm will not in general find a global optimum, its computational advantages can be substantial, and these computational advantages can offset the increased costs of using a suboptimal control rule.

As an example, consider a 20 dimensional system with 10 inputs that models a flexible beam. The global cost matrices E and F are identity matrices of the appropriate sizes. The Q-function for a global control rule is a symmetric 30 by 30 matrix, containing 465 independent parameters. Using RLS to learn the global Q-function would require $O(465^2)$ operations at every time step. On the other hand, suppose we are using 10 independent adaptive controllers. The partial state vector sensed by the i^{th} controller consists of the displacement and velocity of the i^{th} node. The control signal emitted by the i^{th} controller is the acceleration to be applied to the i^{th} node. It includes an exploratory component. The local cost matrices E and F are identity matrices of the appropriate sizes to match the locally sensed state vector and the local control vector. Figure 5.15 illustrates this situation. Each local Q-function is then a symmetric 3 by 3 matrix with 6 independent parameters. Using RLS to learn the Q-function would require $O(6^2)$ operations at every time step at every node. Since there are 10 nodes, the total cost would be $O(10 \times 6^2)$ operations at every time step. This is a reduction of 99.83% in the total computational cost per time step! Table 5.1 shows further comparisons between the costs per time step of the two algorithms for larger systems of the same type. As applied here, each of the independent controllers is of the same size. As the state and action spaces grow, new controllers of the same size are added. If we follow this scheme in general, the cost for the distributed algorithm scales linearly in n, the length of the global state vector,

```
1   For each controller, initialize parameters $\hat{\theta}_1(0)$.
2   $t = 0$, $k = 1$.
3   **repeat** forever {
4       For each controller, initialize $P_k(0)$.
5       **for** $i = 1$ to N {
6           For each controller, determine the local control signal $u_t = U_k x_t + e_t$,
            where $U_k$ is the local controller, $x_t$ is the local state, and $e_t$ is the local
            "exploration" component of the control signal.
7           Apply all of the local control signals to the system simultaneously,
            resulting in a global state transition.
8           For each controller, update the estimates of the $Q$-function parameters,
            $\hat{\theta}_k(i)$, using RLS (Equations (5.14)).
9           $t = t + 1$.
        }
10      **for** each controller {
11          Find the symmetric matrix $\hat{H}_k$ that corresponds to the parameter vector
            $\hat{\theta}_k$.
12          Perform policy improvement based on $\hat{H}_k$: $U_{k+1} = -\hat{H}^{-1}_{k(22)}\hat{H}_{k(21)}$.
13          Initialize parameters $\hat{\theta}_{k+1}(0) = \hat{\theta}_k$.
        }
14      $k = k + 1$
15  }
```

Figure 5.14 The distributed policy iteration algorithm based on Q-learning. The algorithm starts with some stabilizing controller U_0 partitioned among the distributed controllers. t is the total number of time steps. k keeps track of the number of policy iteration steps. i counts the number of time steps since the last change of policy. When $i = N$, one policy improvement step is executed in parallel for all controllers.

The cost per time step of the centralized algorithm is $O((n + m)^4)$, where n is the length of the global state vector, and m is the length of the global control vector[2].

Figure 5.16 demonstrates the performance of the distributed adaptive policy iteration algorithm on the system illustrated in Figure 5.15. The initial state of the system is a random point in a neighborhood around $0 \in \mathbb{R}^{20}$. Panel A of the figure shows that the distributed control rule changes very slowly after the first few policy iteration steps. Panel B shows that the distributed controller remains stationary with respect to the centralized optimal controller after a few policy iteration steps. The upper curve shows the Euclidean distance between the two control rules, and the lower curve shows the max norm of the difference between the two rules. We performed an experiment to assess the increase in control cost associated with using this suboptimal

[2]Following the change of variables (Section 5.3), the vector $\overline{[\ x, u\ ]}$ has $p = (n+m)(n+m+1)/2$ elements. RLS then costs $O(p^2)$ in space and computation at each time step.

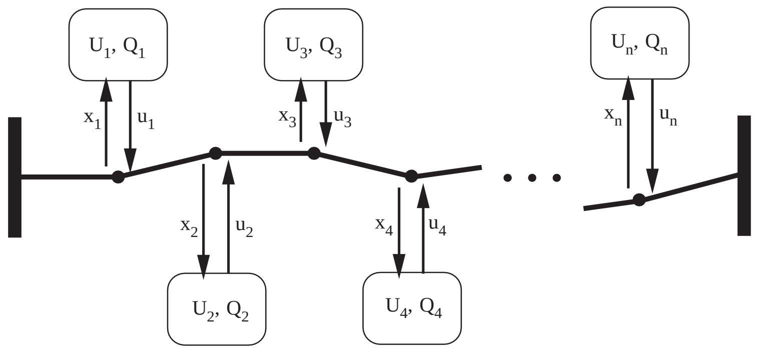


Figure 5.15 Application of the distributed adaptive policy iteration algorithm to a flexible beam. The beam system is a 20-dimensional discrete-time approximation of an Euler-Bernoulli flexible beam supported at both ends. The partial state vector sensed by the i^{th} controller consists of the displacement and velocity of the i^{th} node. The control signal emitted by the i^{th} controller is the acceleration to be applied to the i^{th} node.

controller. We chose 10^6 points from a spherical Gaussian distribution with variance 1.0 about the origin in state space. From each of these points we measured the long-term cost of following the optimal policy, and the long-term cost of following the final distributed policy. The average cost of the optimal policy was 56.26. The average cost of the distributed policy was 60.72, a cost gain of 7.92%. Depending on the system requirements, the decreased computational burden of deriving the distributed controller may be enough to offset the gain in control cost with respect to the optimal cost.

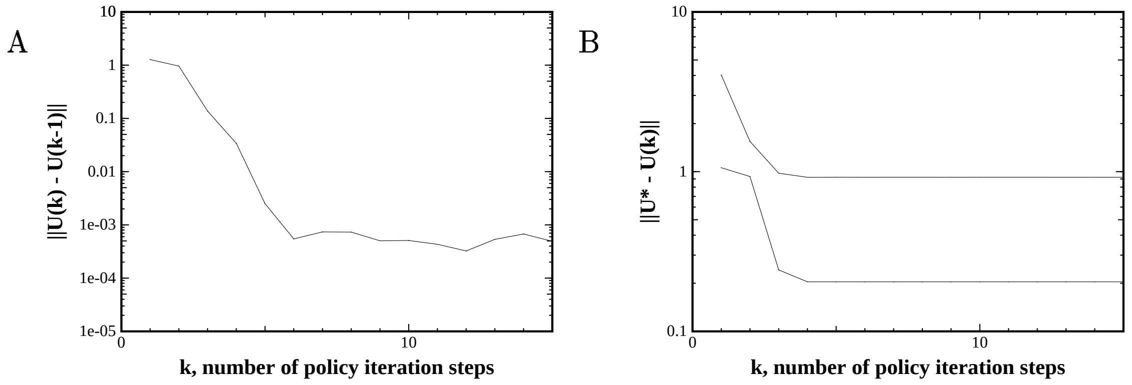


Figure 5.16 Performance of the distributed adaptive policy iteration algorithm. The demonstration system is a simulation of a flexible beam.

Table 5.1 Cost comparisons between the centralized and distributed adaptive policy iteration algorithms. The systems used for the comparison are flexible beams like that in Figure 5.15. The comparisons are between the cost per time step of of the adaptive policy iteration algorithm (Figure 5.1) and the distributed adaptive policy iteration algorithm (Figure 5.14).

Algorithm	*Number of dimensions*	*Number of inputs*	*Number of parameters in Q-function*	*Cost per time step (using RLS)*
Centralized	100	50	$11,325$	$O(11,325^2)$
Distributed			50×6	$O(50 \times 6^2)$
Centralized	200	100	$54,150$	$O(54,150^2)$
Distributed			100×6	$O(100 \times 6^2)$
Centralized	400	200	$180,300$	$O(180,300^2)$
Distributed			200×6	$O(200 \times 6^2)$

5.10 Dealing with a Stochastic System

So far we have only considered deterministic systems. Consider now the application of the adaptive policy iteration algorithm to a system of the form

$$x_{t+1} = Ax_t + Bu_t + \eta_t, \tag{5.45}$$

with feedback control given, as throughout this chapter, by Equation (5.2), and where each η_t is generated independently from an n-dimensional, zero-mean, Gaussian distribution. Instead of trying to solve this problem in general, we will work with a simple 1-dimensional example:

$$\begin{aligned} x_{t+1} &= x_t + u_t + \eta_t \\ u_t &= \alpha x_t. \end{aligned}$$

Using the definition of ϕ_t from Equation (5.13)

$$\begin{aligned}
\phi_t &= \begin{bmatrix} x_t^2 \\ x_t u_t \\ u_t^2 \end{bmatrix} - \gamma \begin{bmatrix} x_{t+1}^2 \\ x_{t+1}u_{t+1} \\ u_{t+1}^2 \end{bmatrix} \\
&= \begin{bmatrix} x_t^2 \\ x_t u_t \\ u_t^2 \end{bmatrix} - \gamma \begin{bmatrix} (x_t + u_t + \eta_t)^2 \\ \alpha(x_t + u_t + \eta_t)^2 \\ \alpha^2(x_t + u_t + \eta_t)^2 \end{bmatrix} \\
&= \begin{bmatrix} x_t^2 \\ x_t u_t \\ u_t^2 \end{bmatrix} - \gamma \begin{bmatrix} (x_t + u_t)^2 + 2\eta_t(x_t + u_t) + \eta_t^2 \\ \alpha(x_t + u_t)^2 + 2\alpha\eta_t(x_t + u_t) + \alpha\eta_t^2 \\ \alpha^2(x_t + u_t)^2 + 2\alpha^2\eta_t(x_t + u_t) + \alpha^2\eta_t^2 \end{bmatrix}
\end{aligned}$$

$$= \begin{bmatrix} x_t^2 \\ x_t u_t \\ u_t^2 \end{bmatrix} - \gamma \begin{bmatrix} (x_t+u_t)^2 \\ \alpha(x_t+u_t)^2 \\ \alpha^2(x_t+u_t)^2 \end{bmatrix} - 2\gamma\eta_t \begin{bmatrix} (x_t+u_t) \\ \alpha(x_t+u_t) \\ \alpha^2(x_t+u_t) \end{bmatrix} - \gamma\eta_t^2 \begin{bmatrix} 1 \\ \alpha \\ \alpha^2 \end{bmatrix}. \quad (5.46)$$

Remembering the discussion of linear Least Squares function approximation from Section 4.5.1, ϕ_t is the noisy input observation. The first two terms in Equation (5.46) come from the deterministic part of the system, and are all that we would get if the state transitions were deterministic. The first two terms are the true input. The other two terms correspond to the input observation noise. The presence of the input observation noise introduces a bias into the RLS estimation of the Q-function parameters. In Section 4.5.2 the input observation noise was zero mean, and we were able to find an instrumental variable with which it was uncorrelated. This allowed the development of the LS TD and RLS TD algorithms for finite-state, finite-action MDPs. But for this LQ problem, the input observation noise is not zero-mean, nor is there readily apparent any instrumental variable with which it is uncorrelated. The deleterious effects of this input observation noise will similarly afflict stochastic approximation methods such as TD(λ). Therefore we are unable to remove the estimation bias caused by the stochastic state transitions.

Based on this analysis, we can conclude that while the adaptive policy iteration algorithm may be applied to a stochastic problem, and it may converge to a stabilizing policy, this policy will always be bounded away from the optimal policy. Furthermore, since the input observation noise depends on the variance of η_t[3], if the variance is large enough, then the bias it introduces may be large enough to cause the adaptive policy iteration algorithm to diverge.

5.11 Conclusions

In this chapter we have taken a first step toward extending the theory of DP-based reinforcement learning to domains with continuous state and action spaces, and to algorithms that use non-linear function approximators. We have concentrated on the problem of Linear Quadratic Regulation. We described a policy iteration algorithm for LQR problems that is proven to converge to the optimal policy. In contrast to standard methods of policy iteration, it does not require a system model; it only requires a suitably accurate estimate of H_{U_k}. This is the first result of which we are aware that proves convergence of a DP-based reinforcement learning algorithm in a domain with continuous states and actions.

The convergence proof for the adaptive policy iteration algorithm requires exact matching between the form of the Q-function for LQR problems and the form of the function approximator used to learn that function, *i.e.*, it requires the function approximator to be quadratic in the input (the state and action vectors). We are able to use linear RLS to train the function approximator through a change of input representation (Section 5.3). Future work will explore the convergence of DP-based reinforcement learning algorithms when applied to non-linear systems for

[3]For multidimensional systems, the input observation noise will depend on the covariance matrix of η_t.

which the form of the Q-function is unknown. It will be necessary in such cases to use more general function approximation techniques, such as multilayer perceptrons. De Lamothe [24], prompted by the work described here, presents some preliminary experimental work along these lines.

It is a common practice to apply DP-based reinforcement learning algorithms to situations that are not covered by any theory. Although these applications have met with some success, one wonders about the possibility of failure. We showed that it is possible for a naive translation of the optimizing Q-learning algorithm to the LQR domain to fail by converging to a destabilizing controller.

We show that the adaptive policy iteration algorithm can be adapted very readily to control the input/output behavior of a system with hidden state. This is a useful development, because it will seldom be the case that one will know the order of an unknown system, or whether the observed output vectors from the system actually contain all of the necessary state information. Overestimating the order of the underlying system does not interfere with convergence or stability as long as the conditions of Theorem 5.1 are satisfied. However, as demonstrated by a simple example, the state space system $\{\tilde{A}, \tilde{B}\}$ corresponding to a non-minimum order ARMAX model of the underlying system may not be controllable. Despite this, the adaptive policy iteration algorithm seems to converge to a unique set point. A theoretical understanding of the convergence of the adaptive policy iteration algorithm despite the violation of controllability is a topic for future work.

The distributed form of the algorithm can yield significant computational savings per time step. The distributed algorithm lacks a convergence theorem, however. It may be possible to find ties between extensions of this work and robust control theory.

The problems that arise when applying the adaptive policy iteration algorithm to a stochastic problem are disheartening. Very few systems of interest will be deterministic. It may be that the only way around these problems is to build a system model. However, once we have a system model, why not use it as is conventionally done, to solve directly for the optimal controller? This is a topic for future research.

CHAPTER 6

CONCLUSIONS

The work presented in this dissertation addresses a number of important issues in the theory and practice of IDP algorithms. These issues, as described in the introduction, are:

1. the question of on-line convergence;
2. whether model-based or model-free methods should be used;
3. the theoretical understanding of the application of an IDP method to a continuous problem;
4. the problem of using a compact function approximator; and
5. the problem of hidden state information.

The results presented in this dissertation have a number of practical implications for the use of IDP algorithms. First, we show in Chapter 3 that IDP algorithms can be used in certain instances to solve MDPs more cheaply than traditional DP algorithms. Next, in Chapter 4 we present new algorithms for TD learning that are more stable, have fewer algorithm parameters to set, and which converge more rapidly in general. Finally, in Chapter 5 we demonstrate one of the pitfalls that can arise when applying IDP algorithms to continuous domains, and we develop a rapidly convergent algorithm that avoids that pitfall. The contributions made in this dissertation are described in more detail below.

Chapter 3 described several new on-line IDP algorithms, RTDP, ARTDP, and IDRTDP, for solving stochastic shortest path problems. These algorithms can be viewed as generalizations of Korf's LRTA*. Using the theory of asynchronous DP, we proved that RTDP and IDRTDP will converge with probability 1 to a proper partial policy with respect to the start set. One of the interesting features of these algorithms is that they tend to concentrate computational resources on the set of relevant states. It is possible for them to converge without ever visiting some of the irrelevant states. This is in contrast to the traditional DP methods like value iteration, which operates by sweeping through the entire state space, updating (backing up) the value of every state as it goes. Our experiments on two instances of the racetrack problem demonstrated that this theoretical advantage of the IDP algorithms could be realized in practice, with both RTDP and ARTDP requiring fewer backups to achieve effective convergence than value iteration. RTDP and ARTDP are both model-based methods, the difference being that RTDP is given a system model to start with, while ARTDP must construct its system model on-line. This retarded the convergence of

ARTDP in our experiments, but it still outperformed the model-free RTQ algorithm by a wide margin.

Chapter 4 described several new on-line IDP algorithms for finding the value functions of both ergodic and absorbing Markov chains. The NTD(λ) algorithm is a normalized version of Sutton's TD(λ). The normalization tends to stabilize the algorithm's performance, reducing its sensitivity to both the state representations and the settings of the control parameters. Using recent results on the convergence of Q-learning, we proved that TD(0) and NTD(0) converge with probability 1 when using a linear function approximator and linearly independent state representations. We then derived the algorithm LS TD using Least Squares and instrumental variables techniques, and proved its probability 1 convergence when using a linear function approximator and linearly independent state representations. RLS TD is a recursive version of LS TD that requires somewhat more restrictive conditions to ensure convergence. We defined the concept of the *TD error variance*, σ_{TD}, and showed in a series of experiments that the convergence rates for NTD(λ) and RLS TD depended linearly on σ_{TD}. RLS TD requires $O(n^2)$ operations per state transition to update the value function estimate. NTD(λ) requires $O(n)$ operations per state transition. Despite this increase in immediate cost, RLS TD should be preferred over TD(λ) or NTD(λ) for problems of moderate size, since it converges faster and it has no control parameters to adjust. The cost of storing an $n \times n$ matrix will become prohibitive for large problems, and it would become necessary to use an $O(n)$ algorithm.

Chapter 5 develops an on-line adaptive policy iteration algorithm based on Q-learning, and applies it to the problem of Linear Quadratic Regulation. We prove that this algorithm convergences under the appropriate conditions to the optimal policy. This is the first work of which we are aware that proves convergence for an IDP algorithm when applied to a continuous domain. This is also the first instance of proven convergence for an IDP algorithm that uses a compact function approximator, that is non-linear in the inputs. We also demonstrate that it is possible for IDP algorithms to fail when applied to a continuous domain, even though they are proven to converge for finite-state, finite-action domains and lookup-table function approximators. In comparison to standard, model-based techniques for finding the optimal LQ policy, we show that the model-free adaptive policy iteration algorithm requires fewer parameters whenever the sizes of the state transition matrices A and B satisfy a certain constraint. We then show how to apply the adaptive policy iteration algorithm to problems in which there is hidden state information, and demonstrate experimentally that overestimating the order of the hidden state does not interfere with convergence. We also describe a heuristic, distributed version of the adaptive policy iteration algorithm that is applicable to systems with loosely coupled subsystems. The distributed algorithm reduces the computational load per time step, but at the cost of reduced performance with respect to the centralized optimal policy.

APPENDIX A

PROOF OF THEOREMS FROM CHAPTER 3

A.1 Proof of Theorem 3.3

Here we prove Theorem 3.3, which extends Korf's [48] convergence theorem for LRTA* to Trial-Based RTDP applied to undiscounted stochastic shortest path problems.

Proof: We first prove the theorem for the special case in which only the cost of the current state is backed up at each time interval, i.e., $B_k = \{x_k\}$ for $k = 0, 1, \ldots$ (see Section 3.4). We then observe that the proof does not change when each B_k is allowed to be an arbitrary set containing x_k. Let $\mathcal{G}$ denote the goal set and let x_k, a_k, and V_k respectively denote the state, action, and evaluation function at time step k in an arbitrary infinite sequence of states, actions, and evaluation functions generated by Trial-Based RTDP starting from an arbitrary start state.

First observe that the evaluation functions remain non-overestimating, i.e., at any time k, $V_k(x) \leq V^*(x)$ for all states x. This is true by induction because $V_{k+1}(x) = V_k(x)$ for all $x \neq x_k$ and if $V_k(x) \leq V^*(x)$ for all $x \in S$, then for all k

$$\begin{aligned} V_{k+1}(x_k) &= \min_{a \in \mathcal{A}(x_k)} \left[R(x_k, a) + \sum_{y \in \mathcal{X}} P(x_k, y, a) V_k(y) \right] \\ &\leq \min_{a \in \mathcal{A}(x_k)} \left[R(x_k, a) + \sum_{y \in \mathcal{X}} P(x_k, y, a) V^*(y) \right] \\ &= V^*(x_k), \end{aligned}$$

where the last equality restates the Bellman Optimality Equation, Equation (2.3).

Let $\mathcal{I} \subseteq \mathcal{X}$ be the set of all states that appear infinitely often in this arbitrary sequence; $\mathcal{I}$ must be nonempty because the state set is finite. Let $\mathcal{A}_{\mathrm{G}}(x) \in \mathcal{A}(x)$ be the set of admissible actions for state x that have zero probability of causing a transition to a state not in $\mathcal{I}$, i.e., $\mathcal{A}_{\mathrm{G}}(x)$ is the set of all actions $a \in \mathcal{A}(x)$ such that $P(x, y, a) = 0$ for all $y \in (\mathcal{X} - \mathcal{I})$. Because states in $\mathcal{X} - \mathcal{I}$ appear a finite number of times, there is a finite time T_0 after which all states visited are in $\mathcal{I}$. Then, with probability one, any action chosen an infinite number of times for any state x that occurs after T_0 must be in $\mathcal{A}_{\mathrm{G}}(x)$ (or else with probability one a transition out of $\mathcal{I}$ would occur), and so with probability one there must exist a time $T_1 \geq T_0$ such that for all $k > T_1$, we not only have that $x_k \in \mathcal{I}$ but also that $a_k \in \mathcal{A}_{\mathrm{G}}(x_k)$.

We know that at each time step k, RTDP backs up the cost of x_k because $x_k \in B_k$. We can write the back-up operation as follows:

$$V_{k+1}(x_k) = \min_{a \in \mathcal{A}(x_k)} \left[R(x_k, a) + \sum_{y \in \mathcal{I}} P(x_k, y, a) V_k(y) + \sum_{y \in (\mathcal{X} - \mathcal{I})} P(x_k, y, a) V_k(y) \right]. \quad \text{(A.1)}$$

But for all $k > T_1$, we know that $x_k \in \mathcal{I}$ and that $P(x_k, y, a_k) = 0$ for all $y \in \mathcal{X} - \mathcal{I}$ because $a_k \in \mathcal{A}_{\text{G}}(x_k)$. Thus, for $k > T_1$ the right-most summation in Equation A.1 is zero. This means that the costs of the states in $\mathcal{X} - \mathcal{I}$ have no influence on the operation of RTDP after T_1. Thus, after T_1, RTDP performs asynchronous DP on a Markovian decision problem with state set $\mathcal{I}$.

If no goal states are contained in $\mathcal{I}$, then all the immediate costs in this Markovian decision problem are positive. Because there is no discounting, it can be shown that asynchronous DP must cause the costs of the states in $\mathcal{I}$ to grow without bound. But this contradicts the fact that the cost of a state can never overestimate its optimal cost, which must be finite due to the existence of a proper policy. Thus $\mathcal{I}$ contains a goal state with probability one.

After T_1, therefore, Trial-Based RTDP performs asynchronous DP on a stochastic shortest path problem with state set $\mathcal{I}$ that satisfies the conditions of the convergence theorem for asynchronous DP applied to undiscounted stochastic shortest path problems (Bertsekas and Tsitsiklis [7], Proposition 3.3, p. 318). Consequently, Trial-Based RTDP converges to the optimal evaluation function of this stochastic shortest path problem. We also know that the optimal evaluation function for this problem is identical to the optimal evaluation function for the original problem restricted to the states in $\mathcal{I}$ because the costs of the states in $\mathcal{X} - \mathcal{I}$ have no influence on the costs of states in $\mathcal{I}$ after time T_1.

Furthermore, with probability one $\mathcal{I}$ contains the set of all states reachable from any start state via any optimal policy. Clearly, $\mathcal{I}$ contains all the start states because each start state begins an infinite number of trails. Trial-Based RTDP always executes a greedy action with respect to the current evaluation function and breaks ties in such a way that it continues to execute all the greedy actions. Because we know that the number of policies is finite and that Trial-Based RTDP converges to the optimal evaluation function restricted to $\mathcal{I}$, there is a time after which it continues to select all the actions that are greedy with respect to the optimal evaluation function, i.e., all the optimal actions. Thus with probability one, $\mathcal{I}$ contains all the states reachable from any start state via any optimal policy, and there is a time after which a controller using RTDP will only execute optimal actions.

Finally, with trivial revision the the above argument holds if RTDP backs up the costs of states other than the current state at each time step, i.e., if each B_k is an arbitrary subset of $\mathcal{X}$.
Q.E.D.

A.2 Proof of Theorem 3.4

Here we prove Theorem 3.4, which states conditions under which IDRTDP will converge when applied to undiscounted stochastic shortest path problems. The proof is nearly the same as that for Trial-based RTDP.

Proof: We first prove the theorem for the special case in which only the cost of the current state is backed up at each time interval, i.e., $B_k = \{x_k\}$ for $k = 0, 1, \ldots$ (see Section 3.9). We then observe that the proof does not change when each B_k is allowed to be an arbitrary set containing x_k. Let $\mathcal{G}$ denote the goal set, $\mathcal{S}$ denote the start set, and let x_k, a_k, and V_k respectively denote the state, action, and evaluation function at time step k in an arbitrary infinite sequence of states, actions, and evaluation functions generated by IDRTDP starting from an arbitrary start state.

First observe that the evaluation functions remain non-overestimating, i.e., at any time k, $V_k(x) \leq V^*(x)$ for all states x. This is true by induction because $V_{k+1}(x) = V_k(x)$ for all $x \neq x_k$ and if $V_k(x) \leq V^*(x)$ for all $x \in S$, then for all k

$$\begin{aligned} V_{k+1}(x_k) &= \min_{a \in \mathcal{A}(x_k)} \left[R(x_k, a) + \sum_{y \in \mathcal{X}} P(x_k, y, a) V_k(y) \right] \\ &\leq \min_{a \in \mathcal{A}(x_k)} \left[R(x_k, a) + \sum_{y \in \mathcal{X}} P(x_k, y, a) V^*(y) \right] \\ &= V^*(x_k), \end{aligned}$$

where the last equality restates the Bellman Optimality Equation, Equation (2.3). Remember that $V^*(x) = \infty$ for any trap state x.

Let $\mathcal{I} \subseteq \mathcal{X}$ be the set of all states that appear infinitely often in this arbitrary sequence; $\mathcal{I}$ must be nonempty because the state set is finite. Let $\mathcal{A}_G(x) \in \mathcal{A}(x)$ be the set of admissible actions for state x that have zero probability of causing a transition to a state not in $\mathcal{I}$, i.e., $\mathcal{A}_G(x)$ is the set of all actions $a \in \mathcal{A}(x)$ such that $P(x, y, a) = 0$ for all $y \in (\mathcal{X} - \mathcal{I})$. Because states in $\mathcal{X} - \mathcal{I}$ appear a finite number of times, there is a finite time T_0 after which all states visited are in $\mathcal{I}$. Then, with probability one, any action chosen an infinite number of times for any state x that occurs after T_0 must be in $\mathcal{A}_G(x)$ (or else with probability one a transition out of $\mathcal{I}$ would occur), and so with probability one there must exist a time $T_1 \geq T_0$ such that for all $k > T_1$, we not only have that $x_k \in \mathcal{I}$ but also that $a_k \in \mathcal{A}_G(x_k)$.

We know that at each time step k, IDRTDP backs up the cost of x_k because $x_k \in B_k$. We can write the back-up operation as follows:

$$V_{k+1}(x_k) = \min_{a \in \mathcal{A}(x_k)} \left[R(x_k, a) + \sum_{y \in \mathcal{I}} P(x_k, y, a) V_k(y) + \sum_{y \in (\mathcal{X} - \mathcal{I})} P(x_k, y, a) V_k(y) \right]. \quad (A.2)$$

But for all $k > T_1$, we know that $x_k \in \mathcal{I}$ and that $P(x_k, y, a_k) = 0$ for all $y \in \mathcal{X} - \mathcal{I}$ because $a_k \in \mathcal{A}_G(x_k)$. Thus, for $k > T_1$ the right-most summation in Equation A.2 is zero. This means that the costs of the states in $\mathcal{X} - \mathcal{I}$ have no influence on the operation of IDRTDP after T_1. Thus, after T_1, IDRTDP performs asynchronous DP on a Markovian decision problem with state set $\mathcal{I}$.

Now, what state are in $\mathcal{I}$? The start states are in $\mathcal{I}$, $\mathcal{S} \subseteq \mathcal{I}$, because of the iterative-deepening time-out provision of IDRTDP. Now, let us suppose that at least one trap state, z, is in $\mathcal{I}$. Because there is no discounting, and all costs not involving goal states are positive, the value estimate for z grows without bound as it is updated infinitely often. There causes no immediate problem, as $V^*(z) = \infty$. But the unbounded growth of $V_k(z)$ means that the value estimates of every state on a path from $\mathcal{S}$ to z will also grow without bound, including the value estimates for the start states. This is a contradiction to the fact that the cost of a state can never overestimate its optimal cost, and the optimal cost of a start state must be finite due to the existance of a partial proper policy with respect to the start states. Therefore, with probability one, no trap states can be elements of $\mathcal{I}$. Next, suppose that no goal state are contained in $\mathcal{I}$. This means that all of the immediate costs in this Markovian decision problem are positive. Because there is no discounting, it can be shown that asynchronous DP must cause the costs of the states in $\mathcal{I}$ to grow without bound. But this is again a contradiction to the fact that the cost of a state can never overestimate its optimal cost, and the optimal cost of a start state must be finite due to the existance of a partial proper policy with respect to the start states. Thus $\mathcal{I}$ contains at least one goal state with probability one.

After T_1, therefore, IDRTDP performs asynchronous DP on a stochastic shortest path problem with state set $\mathcal{I}$ that satisfies the conditions of the convergence theorem for asynchronous DP applied to undiscounted stochastic shortest path problems (Bertsekas and Tsitsiklis [7], Proposition 3.3, p. 318). Consequently, IDRTDP converges to the optimal evaluation function of this stochastic shortest path problem. We also know that the optimal evaluation function for this problem is identical to the optimal evaluation function for the original problem restricted to the states in $\mathcal{I}$ because the costs of the states in $\mathcal{X} - \mathcal{I}$ have no influence on the costs of states in $\mathcal{I}$ after time T_1.

Furthermore, with probability one $\mathcal{I}$ contains the set of all states reachable from any start state via any optimal policy. Clearly, $\mathcal{I}$ contains all the start states because each start state begins an infinite number of trails. IDRTDP always executes a greedy action with respect to the current evaluation function and breaks ties in such a way that it continues to execute all the greedy actions. Because we know that the number of policies is finite and that IDRTDP converges to the optimal evaluation function restricted to $\mathcal{I}$, there is a time after which it continues to select all the actions that are greedy with respect to the optimal evaluation function, i.e., all the optimal actions. Thus with probability one, $\mathcal{I}$ contains all the states reachable from any start state via any optimal policy, and there is a time after which a controller using IDRTDP will only execute optimal actions.

Finally, with trivial revision the the above argument holds if IDRTDP backs up the costs of states other than the current state at each time step, i.e., if each B_k is an arbitrary subset of $\mathcal{X}$.
Q.E.D.

APPENDIX B

SIMULATION DETAILS FOR CHAPTER 3

Except for the discount factor γ, which we set to one throughout the simulations, and the sets B_k, which we set to $\{x_k\}$ for all k, RTDP does not involve any parameters. Gauss-Seidel DP only requires specifying a state ordering for its sweeps. We selected an ordering without concern for any influence it might have on convergence rate. Both ARTDP and RTQ require exploration during the training trials, which we implemented using Equation (3.3). To generate the data described in Section 3.8.2, we decreased the parameter T with successive moves as follows:

$$T(0) = T_{\text{Max}} \tag{B.1a}$$

$$T(k+1) = T_{\text{Min}} + \beta(T(k) - T_{\text{Min}}), \tag{B.1b}$$

where k is the move number (cumulative over trials), $\beta = 0.992$, $T_{\text{Max}} = 75$, and $T_{\text{Min}} = 0.5$.

RTQ additionally requires sequences of learning rate parameters $\alpha_k(x,a)$ (Equation (2.22)) that satisfy the hypotheses of the Q-learning convergence theorem [92,93]. We define these sequences as follows. Let $\alpha_k(x,a)$ denote the learning rate parameter used when the Q-value of the state-action pair (x,a) is backed up at time step k. Let $\eta_k(x,a)$ be the number of backups performed on the Q-value of (x,a) up to time step k. The learning rate $\alpha_k(x,a)$ is defined as follows:

$$\alpha_k(x,a) = \frac{\alpha_0 \tau}{\tau + \eta_k(x,a)} \tag{B.2}$$

where α_0 is the initial learning rate. We set $\alpha_0 = 0.5$ and $\tau = 300$. This equation implements a *search-then-converge* schedule for each $\alpha_k(x,a)$ as suggested by Darken and Moody [21] and Darken, Chang, and Moody [20]. They argue that such schedules can achieve good performance in stochastic optimization tasks. It can be shown that this schedule satisfies the hypotheses of the Q-learning convergence theorem.

APPENDIX C

PROOF OF THEOREMS FROM CHAPTER 4

C.1 Proofs for Section 4.5.1

Lemma 4.1: *If the correlation matrix $Cor(\omega,\omega)$ is nonsingular and finite, and the output observation noise η_i is uncorrelated with the input observations ω_i, then θ_k as defined by Equation (4.14) converges in probability to θ^*.*
Proof: Replacing ψ_i by $(\omega_i'\theta^* + \eta_i)$ in Equation (4.14), and taking the limit as k goes to infinity gives

$$\lim_{k\to\infty} \theta_k = \lim_{k\to\infty} \left[\frac{1}{k}\sum_{i=1}^{k} \omega_i\omega_i'\right]^{-1} \left[\frac{1}{k}\sum_{i=1}^{k}\omega_i\omega_i'\theta^* + \frac{1}{k}\sum_{i=1}^{k}\omega_i\eta_i\right].$$

Looking at all of the individual terms in this expression, we can see under our assumptions that they tend in probability to finite values. $\frac{1}{k}\sum_{i=1}^{k}\omega_i\omega_i'$ converges in probability to $\mathrm{Cor}(\omega,\omega)$ and $\frac{1}{k}\sum_{i=1}^{k}\omega_i\eta_i$ converges in probability to 0. Therefore we are justified in distributing the limit. This gives us

$$\begin{aligned}\lim_{k\to\infty}\theta_k &= \left[\lim_{k\to\infty}\frac{1}{k}\sum_{i=1}^{k}\omega_i\omega_i'\right]^{-1}\left[\lim_{k\to\infty}\frac{1}{k}\sum_{i=1}^{k}\omega_i\omega_i'\theta^* + \lim_{k\to\infty}\frac{1}{k}\sum_{i=1}^{k}\omega_i\eta_i\right]\\ &= \left[\lim_{k\to\infty}\frac{1}{k}\sum_{i=1}^{k}\omega_i\omega_i'\right]^{-1}\left[\lim_{k\to\infty}\frac{1}{k}\sum_{i=1}^{k}\omega_i\omega_i'\theta^* + 0\right]\\ &= \mathrm{Cor}(\omega,\omega)^{-1}\mathrm{Cor}(\omega,\omega)\theta^*\end{aligned}$$

in probability. But according to our assumption, $\mathrm{Cor}(\omega,\omega)$ is invertible. Therefore, $\lim_{k\to\infty}\theta_k = \theta^*$ in probability.
Q.E.D.
Lemma 4.2: *If the correlation matrix $Cor(\rho,\omega)$ is nonsingular and finite, the correlation matrix $Cor(\rho,\zeta) = 0$, and the output observation noise η_i is uncorrelated with the instrumental variables ρ_i, then θ_k as defined by Equation (4.16) converges in probability to θ^*.*
Proof: Replacing ψ_i by $(\omega_i'\theta^* + \eta_i)$ and $\hat{\omega}_i$ by $(\omega_i + \zeta_i)$ in Equation (4.16) and taking the limit as k approaches infinity yields

$$\begin{aligned}\lim_{k\to\infty}\theta_k &= \lim_{k\to\infty}\left[\frac{1}{k}\sum_{i=1}^{k}\rho_i(\omega_i+\zeta_i)'\right]^{-1}\left[\frac{1}{k}\sum_{i=1}^{k}\rho_i(\omega_i'\theta^*+\eta_i)\right]\\ &= \lim_{k\to\infty}\left[\frac{1}{k}\sum_{i=1}^{k}\rho_i\omega_i' + \frac{1}{k}\sum_{i=1}^{k}\rho_i\zeta_i'\right]^{-1}\left[\frac{1}{k}\sum_{i=1}^{k}\rho_i\omega_i'\theta^* + \frac{1}{k}\sum_{i=1}^{k}\rho_i\eta_i\right].\end{aligned}$$

Looking at all of the individual terms in this expression, we can see under our assumptions that they tend in probability to finite values. $\frac{1}{k}\sum_{i=1}^{k}\rho_i\omega_i'$ converges in probability to $\mathrm{Cor}(\rho, \omega)$, $\frac{1}{k}\sum_{i=1}^{k} \rho_i \zeta_i'$ converges in probability to $\mathrm{Cor}(\rho, \zeta)$, and $\frac{1}{k}\sum_{i=1}^{k} \rho_i \eta_i$ converges in probability to 0. Therefore we are justified in distributing the limit. This gives us

$$\begin{aligned}
\lim_{k\to\infty} \theta_k &= \lim_{k\to\infty} \left[\frac{1}{k}\sum_{i=1}^{k} \rho_i \omega_i' + \frac{1}{k}\sum_{i=1}^{k} \rho_i \zeta_i'\right]^{-1} \left[\frac{1}{k}\sum_{i=1}^{k} \rho_i \omega_i' \theta^* + \frac{1}{k}\sum_{i=1}^{k} \rho_i \eta_i\right] \\
&= \left[\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \rho_i \omega_i' + \lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \rho_i \zeta_i'\right]^{-1} \left[\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \rho_i \omega_i' \theta^* + \lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \rho_i \eta_i\right] \\
&= \left[\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \rho_i \omega_i' + 0\right]^{-1} \left[\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \rho_i \omega_i' \theta^* + 0\right] \\
&= \mathrm{Cor}(\rho, \omega)^{-1} \mathrm{Cor}(\rho, \omega) \theta^*
\end{aligned}$$

in probability. But according to our assumption, $\mathrm{Cor}(\rho, \omega)$ is invertible. Therefore, $\lim_{k\to\infty} \theta_k = \theta^*$ in probability.
Q.E.D.

C.2 Proofs for Section 4.5.2

First we'll examine $\sum_{y\in\mathcal{X}} P(x,y)(R(x,y) - \bar{R}_x)$ for an arbitrary state x.

$$\begin{aligned}
\sum_{y\in\mathcal{X}} P(x,y)(R(x,y) - \bar{R}_x) &= \sum_{y\in\mathcal{X}} P(x,y)R(x,y) - \sum_{y\in\mathcal{X}} P(x,y)\bar{R}_x \\
&= \sum_{y\in\mathcal{X}} P(x,y)R(x,y) - \bar{R}_x \\
&= \bar{R}_x - \bar{R}_x \\
&= 0.
\end{aligned}$$

Now we're ready to prove the stated lemmas.
Lemma 4.3: *For any Markov chain, if x and y are states such that $P(x,y) > 0$, with $\eta_{xy} = R(x,y) - \bar{R}_x$ and $\omega_x = (\phi_x - \gamma \sum_{y\in\mathcal{X}} P(x,y)\phi_y)$, then $\mathcal{E}\{\eta\} = 0$, and Cor$(\omega, \eta) = 0$.*
Proof: The result in the preceeding paragraph leads directly to a proof that $\mathcal{E}\{\eta\} = 0$,

$$\begin{aligned}
\mathcal{E}\{\eta\} &= \mathcal{E}\left\{R(x,y) - \bar{R}_x\right\} \\
&= \sum_{x\in\mathcal{X}} \pi_x \sum_{y\in\mathcal{X}} P(x,y)(R(x,y) - \bar{R}_x) \\
&= \sum_{x\in\mathcal{X}} \pi_x \cdot 0 \\
&= 0,
\end{aligned}$$

and to a proof that $\mathrm{Cor}(\omega, \eta) = 0$,

$$\mathrm{Cor}(\omega, \eta) = \mathcal{E}\{\omega\eta\}$$

$$
\begin{aligned}
&= \sum_{x\in\mathcal{X}} \pi_x \sum_{y\in\mathcal{X}} P(x,y)\,[\omega_x \eta_{xy}] \\
&= \sum_{x\in\mathcal{X}} \pi_x \sum_{y\in\mathcal{X}} P(x,y)\omega_x(R(x,y) - \bar{R}_x) \\
&= \sum_{x\in\mathcal{X}} \pi_x\omega_x \sum_{y\in\mathcal{X}} P(x,y)(R(x,y) - \bar{R}_x) \\
&= \sum_{x\in\mathcal{X}} \pi_x\omega_x \cdot 0 \\
&= 0.
\end{aligned}
$$

Q.E.D.

Lemma 4.4: *For any Markov chain, if (1) x and y are states such that $P(x,y) > 0$; (2) $\zeta_{xy} = \gamma \sum_{z\in\mathcal{X}} P(x,z)\phi_z - \gamma\phi_y$; (3) $\eta_{xy} = R(x,y) - \bar{R}_x$; and (4) $\rho_x = \phi_x$, then (1) Cor$(\rho,\eta) = 0$; and (2) Cor$(\rho,\zeta) = 0$.*

Proof: First we will consider $\mathrm{Cor}(\rho,\eta)$.

$$
\begin{aligned}
\mathrm{Cor}(\rho,\eta) &= \mathcal{E}\{\rho\eta'\} \\
&= \sum_{x\in\mathcal{X}} \pi_x \sum_{y\in\mathcal{X}} P(x,y)\left[\rho_x \eta'_{xy}\right] \\
&= \sum_{x\in\mathcal{X}} \pi_x \sum_{y\in\mathcal{X}} P(x,y)\phi_x(R(x,y) - \bar{R}_x)' \\
&= \sum_{x\in\mathcal{X}} \pi_x\phi_x \sum_{y\in\mathcal{X}} P(x,y)(R(x,y) - \bar{R}_x)' \\
&= \sum_{x\in\mathcal{X}} \pi_x\phi_x \cdot 0 \\
&= 0.
\end{aligned}
$$

Now for $\mathrm{Cor}(\rho,\zeta)$.

$$
\begin{aligned}
\mathrm{Cor}(\rho,\zeta) &= \mathcal{E}\{\rho\zeta'\} \\
&= \sum_{x\in\mathcal{X}} \pi_x \sum_{y\in\mathcal{X}} P(x,y)\left[\rho_x \zeta'_{xy}\right] \\
&= \sum_{x\in\mathcal{X}} \pi_x \sum_{y\in\mathcal{X}} P(x,y)\phi_x(\gamma \sum_{z\in\mathcal{X}} P(x,z)\phi_z - \gamma\phi_y)' \\
&= \sum_{x\in\mathcal{X}} \pi_x\phi_x \sum_{y\in\mathcal{X}} P(x,y)(\gamma \sum_{z\in\mathcal{X}} P(x,z)\phi'_z) - \sum_{x\in\mathcal{X}} \pi_x\phi_x \sum_{y\in\mathcal{X}} P(x,y)\gamma\phi'_y \\
&= \sum_{x\in\mathcal{X}} \pi_x\phi_x\gamma \sum_{z\in\mathcal{X}} P(x,z)\phi'_z - \sum_{x\in\mathcal{X}} \pi_x\phi_x\gamma \sum_{y\in\mathcal{X}} P(x,y)\phi'_y \\
&= 0.
\end{aligned}
$$

Q.E.D.

C.3 Proofs for Section 4.5.3

Lemma 4.5: *For any Markov chain, when (1) θ_k is found using algorithm LS TD; (2) each state $x \in \mathcal{X}$ is visited infinitely often; (3) each state $x \in \mathcal{X}$ is visited in the long run with probability 1 in proportion π_x; and (4) $[\Phi'\Pi(I - \gamma P)\Phi]$ is invertible, then $\theta_{\text{LSTD}} = [\Phi'\Pi(I - \gamma P)\Phi]^{-1} \left[\Phi'\Pi\bar{R}\right]$ with probability 1.*

Proof: Equation 4.21 (repeated here) gives us the k^{th}estimate found by algorithm LS TD for θ^*,

$$\theta_k = \left[\frac{1}{k}\sum_{i=1}^{k} \phi_i(\phi_i - \gamma\phi_{i+1})'\right]^{-1} \left[\frac{1}{k}\sum_{i=1}^{k} \phi_i R_i\right].$$

As k grows we have by condition (2) that the *sampled* transition probabilities between each pair of states approaches the *true* transition probabilities, P, with probability 1. We also have by condition (3) that each state $x \in \mathcal{X}$ is visited in the proportion π_x with probability 1. Therefore, given condition (4) we can express the limiting estimate found by algorithm LS TD, θ_{LSTD}, as

$$\begin{aligned}
\theta_{\text{LSTD}} &= \lim_{k\to\infty} \theta_k \\
&= \lim_{k\to\infty} \left[\frac{1}{k}\sum_{i=1}^{k} \phi_i(\phi_i - \gamma\phi_{i+1})'\right]^{-1} \left[\frac{1}{k}\sum_{i=1}^{k} \phi_i R_i\right] \\
&= \left[\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \phi_i(\phi_i - \gamma\phi_{i+1})'\right]^{-1} \left[\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \phi_i R_i\right] \\
&= \left[\sum_x \pi_x \sum_y P(x,y)\phi_x\left(\phi_x - \gamma\phi_y\right)'\right]^{-1} \left[\sum_x \pi_x\phi_x \sum_y P(x,y)R(x,y)\right] \\
&= \left[\sum_x \pi_x \sum_y P(x,y)\phi_x\left(\phi_x - \gamma\phi_y\right)'\right]^{-1} \left[\sum_x \pi_x \bar{R}_x \phi_x\right] \\
&= [\Phi'\Pi(I - \gamma P)\Phi]^{-1} \left[\Phi'\Pi\bar{R}\right].
\end{aligned}$$

Q.E.D.

APPENDIX D

PROOFS OF LEMMAS FROM CHAPTER 5

D.1 Proof of Lemma 5.1

Proof: Lemma 5.1 follows directly from Lemmata D.2 and D.3.
Q.E.D.

Lemmata D.1 through D.3 are presented, along with some prelimiary definitions, in Sections D.1.1 through D.1.4.

D.1.1 Preliminary Definitions

Let U_1 be a stabilizing controller for this system, and let K_1 be the associated cost matrix. Let U_2 be the result of performing the policy improvement algorithm on U_1, *i.e.*,

$$U_2 = -\gamma(F + \gamma B' K_1 B)^{-1} B' K_1 A. \tag{D.1}$$

Let K_2 be the cost matrix associated with U_2. Define $A_1 = A + BU_1$, and $A_2 = A + BU_2$.

We know [6] that the cost matrix K_{U} for a given control matrix U satisfies Equation (5.6), which is repeated here as

$$K_{\mathrm{U}} = E + U'FU + \gamma\left(A + BU\right)' K_{\mathrm{U}} \left(A + BU\right), \tag{D.2}$$

and the equation

$$K_{\mathrm{U}} = \sum_{i=0}^{\infty} \gamma^i (A + BU)^{i\prime} (E + U'FU)(A + BU)^i. \tag{D.3}$$

D.1.2 Lemma D.1

Lemma D.1 *If $\{A, B\}$ is controllable, U_1 is stabilizing, and*
$U_2 = -\gamma(F + \gamma B' K_1 B)^{-1} B' K_1 A$, *then*

$$K_1 - K_2 = \sum_{i=0}^{\infty} \gamma^i A_2^{i\prime} \left[(U_1 - U_2)'(F + \gamma B' K_1 B)(U_1 - U_2)\right] A_2^i,$$

where $A_1 = A + BU_1$ and $A_2 = A + BU_2$.

Proof: First, rewrite K_1 as:

$$\begin{aligned}
K_1 &= K_1 + \sum_{i=1}^{\infty}(\gamma^i A_2^{i\prime} K_1 A_2^i) - \sum_{i=1}^{\infty}(\gamma^i A_2^{i\prime} K_1 A_2^i) \\
&= \sum_{i=0}^{\infty}(\gamma^i A_2^{i\prime} K_1 A_2^i) - \sum_{i=1}^{\infty}(\gamma^i A_2^{i\prime} K_1 A_2^i) \\
&= \sum_{i=0}^{\infty}(\gamma^i A_2^{i\prime} K_1 A_2^i) - \sum_{i=0}^{\infty}(\gamma^i A_2^{i\prime}[\gamma A_2{}' K_1 A_2] A_2^i) \\
&= \sum_{i=0}^{\infty}\gamma^i A_2^{i\prime}[K_1 - \gamma A_2{}' K_1 A_2] A_2^i \qquad \text{(D.4)}
\end{aligned}$$

Combining Equations (D.3) and Equation (D.4) we get

$$K_1 - K_2 = \sum_{i=0}^{\infty}\gamma^i A_2^{i\prime}\left[K_1 - \gamma A_2{}' K_1 A_2 - E - U_2{}' F U_2\right] A_2^i. \qquad \text{(D.5)}$$

Let us define $D = [K_1 - \gamma A_2{}' K_1 A_2 - E - U_2{}' F U_2]$.

Substituting from Equation (D.2) into the definition of D, we get

$$\begin{aligned}
D &= E + U_1{}' F U_1 + \gamma A_1{}' K_1 A_1 - \gamma A_2{}' K_1 A_2 - E - U_2{}' F U_2 \\
&= U_1{}' F U_1 + \gamma A_1{}' K_1 A_1 - \gamma A_2{}' K_1 A_2 - U_2{}' F U_2.
\end{aligned}$$

Expanding $A_2{}' K_1 A_2$ yields

$$\begin{aligned}
D &= U_1{}' F U_1 + \gamma A_1{}' K_1 A_1 \\
&\quad -\gamma A' K_1 A - \gamma A' K_1 B U_2 - \gamma U_2{}' B' K_1 A - \gamma U_2{}' B' K_1 B U_2 \\
&\quad -U_2{}' F U_2 \\
&= U_1{}' F U_1 + \gamma A_1{}' K_1 A_1 - \gamma A' K_1 A \\
&\quad -\gamma A' K_1 B U_2 - \gamma U_2{}' B' K_1 A \\
&\quad -U_2{}'(F + \gamma B' K_1 B) U_2.
\end{aligned}$$

Using the definition of U_2 from Equation (D.1) now gives

$$\begin{aligned}
D &= U_1{}' F U_1 + \gamma A_1{}' K_1 A_1 - \gamma A' K_1 A \\
&\quad +U_2{}'(F + \gamma B' K_1 B) U_2 + U_2{}'(F + \gamma B' K_1 B) U_2 \\
&\quad -U_2{}'(F + \gamma B' K_1 B) U_2 \\
&= U_1{}' F U_1 + \gamma A_1{}' K_1 A_1 - \gamma A' K_1 A + U_2{}'(F + \gamma B' K_1 B) U_2.
\end{aligned}$$

Finally, expanding $A_1{}' K_1 A_1$ and again using the definition of U_2 leads to

$$\begin{aligned}
D &= U_1{}' F U_1 \\
&\quad +\gamma A' K_1 A + \gamma A' K_1 B U_1 + \gamma U_1{}' B' K_1 A + \gamma U_1{}' B' K_1 B U_1 \\
&\quad -\gamma A' K_1 A
\end{aligned}$$

$$
\begin{aligned}
&+U_2{}'(F+\gamma B'K_1B)U_2 \\
&= U_1{}'(F+\gamma B'K_1B)U_1 \\
&\quad +\gamma A'K_1BU_1+\gamma U_1{}'B'K_1A \\
&\quad +U_2{}'(F+\gamma B'K_1B)U_2 \\
&= U_1{}'(F+\gamma B'K_1B)U_1 - U_2{}'(F+\gamma B'K_1B)U_1 \\
&\quad -U_1{}'(F+\gamma B'K_1B)U_2 + U_2{}'(F+\gamma B'K_1B)U_2 \\
&= (U_1-U_2)'(F+\gamma B'K_1B)(U_1-U_2).
\end{aligned}
$$

Substitute this final expression for D back into Equation (D.5) to get the desired result

$$K_1 - K_2 = \sum_{i=0}^{\infty} \gamma^i A_2^{i\prime} \left[(U_1-U_2)'(F+\gamma B'K_1B)(U_1-U_2)\right] A_2^i.$$

Q.E.D.

D.1.3 Lemma D.2

Lemma D.2 *If $\{A,B\}$ is controllable, U_1 is stabilizing, and $U_2 = -\gamma(F+\gamma B'K_1B)^{-1}B'K_1A$, then*

$$\sigma(U_1) - \sigma(U_2) \geq \Delta \|U_1 - U_2\|^2.$$

where $0 < \Delta = \underline{\sigma}(F)$.

Proof: By Lemma D.1, we know that

$$
\begin{aligned}
K_1 - K_2 &= \sum_{i=0}^{\infty} \gamma^i A_2^{i\prime} \left[(U_1-U_2)'(F+\gamma B'K_1B)(U_1-U_2)\right] A_2^i \\
&\geq (U_1-U_2)'F(U_1-U_2),
\end{aligned}
$$

since all of the summands are positive.

Taking the trace of both sides we get

$$
\begin{aligned}
\operatorname{trace}(K_1) - \operatorname{trace}(K_2) &= \operatorname{trace}(K_1 - K_2) \\
&\geq \operatorname{trace}((U_1-U_2)'(F+\gamma B'K_1B)(U_1-U_2)) \\
&\geq \underline{\sigma}(F)\|U_1-U_2\|^2.
\end{aligned}
$$

Noting that F is positive definite and substituting from the definitions of $\sigma(U_1)$ and $\sigma(U_2)$ gives us the final result

$$0 < \sigma(U_1) - \sigma(U_2) \geq \Delta \|U_1 - U_2\|^2.$$

Q.E.D.

D.1.4 Lemma D.3

Lemma D.3 *If* $\{A, B\}$ *is controllable,* U_1 *is stabilizing, and* $U_2 = -\gamma(F + \gamma B'K_1B)^{-1}B'K_1A$*, then*

$$\sigma(U_1) - \sigma(U_2) \leq \delta\|U_1 - U_2\|^2.$$

where $0 < \delta = \text{trace}\,(F + \gamma B'K_1B)\|G\|^2$, $G = \left(\sum_{i=0}^{\infty}\gamma^{(i/2)}A_2^i\right)$, $A_1 = A + BU_1$*, and* $A_2 = A + BU_2$*.*

Proof: By Lemma D.1, we know that

$$\begin{aligned} K_1 - K_2 &= \sum_{i=0}^{\infty}\gamma^i A_2^{i\prime}\left[(U_1 - U_2)'(F + \gamma B'K_1B)(U_1 - U_2)\right]A_2^i \\ &\leq \left(\sum_{i=0}^{\infty}\gamma^{(i/2)}A_2^{i\prime}\right)(U_1 - U_2)'(F + \gamma B'K_1B)(U_1 - U_2)\left(\sum_{i=0}^{\infty}\gamma^{(i/2)}A_2^i\right) \\ &= G'(U_1 - U_2)'(F + \gamma B'K_1B)(U_1 - U_2)G. \end{aligned}$$

Taking the trace of both sides we get

$$\begin{aligned} \text{trace}(K_1) - \text{trace}(K_2) &= \text{trace}(K_1 - K_2) \\ &\leq \text{trace}(G'(U_1 - U_2)'(F + \gamma B'K_1B)(U_1 - U_2)G) \\ &\leq \text{trace}\,(F + \gamma B'K_1B)\|G\|^2\|U_1 - U_2\|^2. \end{aligned}$$

Noting that $(F + \gamma B'K_1B)$ is positive definite and substituting from the definitions of $\sigma(U_1)$ and $\sigma(U_2)$ gives us the final result

$$0 < \sigma(U_1) - \sigma(U_2) \leq \delta\|U_1 - U_2\|^2.$$

Q.E.D.

D.2 Proof of Lemma 5.2

Proof: Let us consider the k^{th} estimation interval. $k_k = \Theta(H_{U_k})$, the true vector of parameters for the function Q_{U_k}. $\hat{\theta}_k = \hat{\theta}_k(N)$ is the estimate of k_k at the end of the k^{th} estimation interval. The parameter estimates are initialized for the k^{th} estimation interval with the final values from the previous estimation interval, *i.e.,* $\hat{\theta}_k(0) = \hat{\theta}_{k-1}$. The RLS algorithm is initialized at the start of the k^{th} estimation interval by setting the inverse covariance matrix $P_k(0) = P_0$, and setting the initial parameter estimates to the final values from the previous interval, *i.e.,* $\hat{\theta}_k(0) = \hat{\theta}_{k-1}$. Define $\tilde{\theta}_k(i) = \hat{\theta}_k(i) - k_k$. Then following Goodwin and Sin [30] we have

$$\tilde{\theta}_k(i) = P_k(i)P_k(i-1)^{-1}\tilde{\theta}_k(i-1)$$

for all $i > 0$. Applying this relation recursively results in

$$\tilde{\theta}_k(i) = P_k(i)P_k(0)^{-1}\tilde{\theta}_k(0).$$

Taking the norms of both sides we have

$$\|\tilde{\theta}_k(i)\| = \|P_k(i)P_k(0)^{-1}\tilde{\theta}_k(0)\|$$

$$\leq \|P_k(i)\| \cdot \|P_k(0)^{-1}\| \cdot \|\tilde{\theta}_k(0)\|. \tag{D.6}$$

Now, $P_k(i)^{-1} = P_k(0)^{-1} + \sum_{i=1}^{N} \phi_k(i)\phi_k(i)'$. Therefore,

$$\begin{aligned} \|P_k(i)^{-1}\| &= \|P_k(0)^{-1} + \textstyle\sum_{i=1}^{N} \phi_k(i)\phi_k(i)'\| \\ &\geq \| \textstyle\sum_{i=1}^{N} \phi_k(i)\phi_k(i)'\| \\ &\geq N\epsilon_0 I \end{aligned} \tag{D.7}$$

We also know that

$$\|P_k(0)^{-1}\| = \frac{1}{p_0}. \tag{D.8}$$

Substituting Equation (D.7) and Equation (D.8) into Equation (D.6) and using the definition of ϵ_N yields

$$\begin{aligned} \|k_k - \hat{\theta}_k(i)\| &= \|\tilde{\theta}_k(i)\| \\ &\leq \epsilon_N \|\tilde{\theta}_k(0)\| \\ &= \epsilon_N \|\hat{\theta}_k(0) - k_k\| \\ &= \epsilon_N \|\hat{\theta}_{k-1} - k_k\| \\ &= \epsilon_N \|\hat{\theta}_{k-1} - k_k + k_{k-1} - k_{k-1}\| \\ &\leq \epsilon_N (\|k_{k-1} - \hat{\theta}_{k-1}\| + \|k_k - k_{k-1}\|), \end{aligned}$$

and we have the desired result.
Q.E.D.

BIBLIOGRAPHY

[1] Anderson, C. W. Strategy learning with multilayer connectionist representations. Technical Report 87-509.3, GTE Laboratories Incorporated, Computer and Intelligent Systems Laboratory, 40 Sylvan Road, Waltham, MA 02254, May 1988.

[2] Aoki, M. *Optimization of Stochastic Systems: Topics in discrete-time dynamics.* Academic Press, Inc., San Diego, 1989.

[3] Barr, A. and Feigenbaum, E. A. *The Handbook of Artificial Intelligence.* William Kaufmann, Inc., Los Altos, CA, 1981.

[4] Barto, A. G., Bradtke, S. J., and Singh, S. P. Learning to act using real-time dynamic programming. *Artificial Intelligence.* Accepted.

[5] Barto, A. G., Sutton, R. S., and Anderson, C. W. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:835–846, 1983.

[6] Bertsekas, D. P. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice Hall, Englewood Cliffs, NJ, 1987.

[7] Bertsekas, D. P. and Tsitsiklis, J. N. *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[8] Bradtke, S. J. Incremental dynamic programming for sequential decision problems: A dissertation proposal. unpublished, December 1990.

[9] Bradtke, S. J. Reinforcement learning applied to linear quadratic regulation. In *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann.

[10] Bradtke, S. J., Ydstie, B. E., and Barto, A. G. Adaptive linear quadratic control using policy iteration. In *Proceedings of the American Control Conference, 1994*.

[11] Bradtke, S. J., Ydstie, B. E., and Barto, A. G. Adaptive linear quadratic control using policy iteration. *IEEE TAC.* Submitted.

[12] Chapman, D. Penquins can make cake. *AI Magazine*, 10:45–50, 1989.

[13] Chapman, D. and Kaelbling, L. P. Learning from delayed reinforcement in a complex domain. Technical Report TR-90-11, Teleos Research, 576 Middlefield Road, Palo Alto, CA 94301, December 1990.

[14] Chapman, D. and Kaelbling, L. P. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of IJCAI*, 1991.

[15] Charniak, E. and McDermott, D. *Introduction to Artificial Intelligence.* Addison-Wesley Publishing Company, Reading, MA, 1985.

[16] Chen, C.-T. *Linear system theory and design.* Holt, Rinehart and Winston, Inc., New York, 1984.

[17] Chrisman, L. Planning for closed-loop execution using partially observable markovian decision processes. In *AAAI Spring Symposium Series: Control of Selective Sensing*, 1992.

[18] Chrisman, L. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI-92 Proceedings. Tenth National Conference on Artificial Intelligence*, 1992.

[19] Christensen, J. and Korf, R. E. A unified theory of heuristic evaluation functions and its application to learning. In *Proceedings of the Fifth National Conference on Artificial Intelligence AAAI-86*, pages 148–152, San Mateo, CA, 1986. Morgan Kaufmann.

[20] Darken, C., Chang, J., and Moody, J. Learning rate schedules for faster stochastic gradient search. In *Neural Networks for Signal Processing 2 — Proceedings of the 1992 IEEE Workshop.* IEEE Press, 1992.

[21] Darken, C. and Moody, J. Note on learning rate schedule for stochastic optimization. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S, editors, *Advances in Neural Information Processing Systems 3*, pages 832–838, San Mateo, CA, 1991. Morgan Kaufmann.

[22] Dayan, P. The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362, 1992.

[23] Dayan, P. and Sejnowski, T. J. Td(λ): Convergence with probability 1. *Machine Learning.* In press.

[24] De Lamothe, P. E. Policy improvement using basis function networks. Master's thesis, University of Massachusetts at Amherst, March 1993.

[25] Dean, T. and Boddy, M. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, 1988.

[26] Denardo, E. V. Contraction mappings in the theory underlying dynamic programming. *SIAM Review*, 9(2):165–177, April 1967.

[27] Gardner, M. Mathematical games. *Scientific American*, 228:108, January 1973.

[28] Gelperin, D. On the optimality of A*. *Artificial Intelligence*, 8:69–76, 1977.

[29] Ginsberg, M. L. Universal planning: An (almost) universally bad idea. *AI Magazine*, 10:40–44, 1989.

[30] Goodwin, G. C. and Sin, K. S. *Adaptive filtering prediction and control.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[31] Gullapalli, V. *Reinforcement Learning and Its Application to Control.* PhD thesis, University of Massachusetts, Amherst, MA, 1992.

[32] Gullapalli, V. and Barto, A. G. Convergence of indirect adaptive asynchronous value iteration algorithms. In Cowan, J. D., Tesauro, G., and Alspector, J, editors, *Advances in Neural Information Processing Systems 6*, San Francisco, CA, 1994. Morgan Kaufmann Publishers.

[33] Howard, R. A. *Dynamic Programming and Markov Processes.* John Wiley & Sons, Inc., New York, 1960.

[34] Ishida, T. Moving target search with intelligence. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Menlo Park, CA, 1992. AAAI Press/MIT Press.

[35] Ishida, T. and Korf, R. E. Moving target search. In *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, 1991.

[36] Jaakkola, T., Jordan, M. I., and Singh, S. P. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation.* Submitted.

[37] Jaakkola, T., Jordan, M. I., and Singh, S. P. Stochastic convergence of iterative dp algorithms. In *Proceedings of the Conference on Neural Information Processing Systems — Natural and Synthetic*, 1993. Accepted.

[38] Jacobson, D. H. and Mayne, D. Q. *Differential Dynamic Programming*, volume 24 of *Modern Analytic and Computational Methods in Science and Mathematics.* American Elsevier Publishing Company, Inc., New York, 1970.

[39] Jalali, A. and Ferguson, M. Computationally efficient adaptive control algorithms for Markov chains. In *Proceedings of the 28th Conference on Decision and Control*, pages 1283–1288, Tampa, FL, December 1989.

[40] Jalali, A. and Ferguson, M. Adaptive control of Markov chains with local updates. *Systems and Control Letters*, 14:209–218, 1990.

[41] Jordan, M. I. and Jacobs, R. A. Learning to control an unstable system with forward modeling. In *Advances in Neural Information Processing Systems 2.* Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[42] Kalman, R. E. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering, Transactions of the ASME, Series D*, 82(1):35–45, 1960.

[43] Kalman, R. E. and Bucy, R. S. New results in linear filtering and prediction theory. *Journal of Basic Engineering, Transactions of the ASME, Series D*, 83(3):95–108, 1961.

[44] Kemeny, J. G. and Snell, J. L. *Finite Markov chains.* Springer-Verlag, New York, 1976.

[45] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[46] Kleinman, D. L. On an iterative technique for Riccati equation computations. *IEEE Transactions on Automatic Control*, pages 114–115, February 1968.

[47] Korf, R. E. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[48] Korf, R. E. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[49] Kwon, W. H. and Pearson, A. E. A modified quadratic cost problem and feedback stabilization of a linear system. *Machine Learning: Proceedings of the Eighth International Workshop*, 22:838–842, 1977.

[50] Larson, R. E. and Casti, J. L. *Principles of Dynamic Programming. Part I: Basic Analytic and Computational Methods*, volume 7 of *Control and Systems Theory.* Marcel Dekker, Inc., New York, 1978.

[51] Lin, L.-J. Self-improvement based on reinforcement learning, planning and teaching. In Birnbaum, L. A. and Collins, G. C, editors, *American Association for Artificial Intelligence*, pages 323–327, San Mateo, CA, 1991. Morgan Kaufmann.

[52] Lin, L.-J. Self-improving reactive agents: Case studies of reinforcement learning frameworks. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 297–305, Cambridge, MA, 1991. MIT Press.

[53] Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[54] Lin, L.-J. and Mitchell, T. M. Memory approaches to reinforcement learning in non-markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, 1992.

[55] Ljung, L. and Söderström, T. *Theory and Practice of Recursive Identification.* MIT Press, Cambridge, MA, 1983.

[56] Luenberger, D. G. *Introduction to dynamic systems: Theory, Models, and Applications.* John Wiley & Sons, New York, 1979.

[57] Lukes, G., Thompson, B., and Werbos, P. J. Expectation driven learning with an associative memory. In *Proceedings of the International Joint Conference on Neural Networks*, pages I:521–524, 1990.

[58] Mayne, D. Q. and Michalska, H. Receding horizon control of nonlinear systems. *Machine Learning: Proceedings of the Eighth International Workshop*, 35:814–824, 1990.

[59] McCallum, R. A. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

[60] Mérõ, L. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23:13–27, 1984.

[61] Moore, A. W. and Atkeson, C. G. Memory-based reinforcement learning: Efficient computation with prioritized sweeping. In *Advances in Neural Information Processing Systems 5*, San Mateo, CA. Morgan Kaufmann.

[62] Moore, A. W. and Atkeson, C. G. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 8, 1992. **VERIFY THIS!**

[63] Musliner, D. J., Durfee, E. H., and Shin, K. G. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence.* Submitted.

[64] Nilsson, N. J. *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, CA, 1980.

[65] Puterman, M. L. and Shin, M. C. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24:1127–1137, 1978.

[66] Robbins, H. and Monro, S. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[67] Ross, S. M. *Introduction to Stochastic Dynamic Programming.* Academic Press, New York, 1983.

[68] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.

[69] Samuel, A. L. Some studies in machine learning using the game of checkers. II—Recent progress. *IBM Journal of Research and Development*, pages 601–617, November 1967.

[70] Sato, M., Abe, K., and Takeda, H. Learning control of finite Markov chains with explicit trade-off between estimation and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 18:677–684, 1988.

[71] Schoppers, M. Building plans to monitor and exploit open-loop and closed-loop dynamics. *Artificial Intelligence.* Submitted.

[72] Schoppers, M. J. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1039–1046, Menlo Park, CA, 1987.

[73] Schoppers, M. J. In defense of reaction plans as caches. *AI Magazine*, 10:51–60, 1989.

[74] Schwartz, A. Doing away with temporal discounting. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, MA, June 1993.

[75] Schwartz, A. Thinking locally to act globally: A novel approach to reinforcement learning. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, 1993.

[76] Singh, S. P. Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning*, 8(3/4):323–339, May 1992.

[77] Singh, S. P. *Learning to Solve Markovian Decision Processes.* PhD thesis, University of Massachusetts, Amherst, MA, 1993.

[78] Singh, S. P. and Yee, R. C. An upper bound on the loss from approximate optimal-value functions. *Machine Learning.* to appear.

[79] Söderström, T. and Stoica, P. G. *Instrumental Variable Methods for System Identification.* Springer-Verlag, Berlin, 1983.

[80] Sofge, D. A. and White, D. A. Neural network based process optimization and control. In *Proceedings of the 29th Conference on Decision and Control*, Honolulu, Hawaii, December 1990.

[81] Sutton, R. S. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, Department of Computer and Information Science, University of Massachusetts at Amherst, Amherst, MA 01003, 1984.

[82] Sutton, R. S. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[83] Sutton, R. S. First results with DYNA, an integrated architecture for learning, planning and reacting. In *Proceedings of the 1990 AAAI Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*, 1990.

[84] Sutton, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Sanderson, A. C., Desrochers, A. A., and Valavanis, K, editors, *Proceedings of the IEEE International Symposium on Intelligent Control*, Albany, New York, 25-26 September 1990. IEEE.

[85] Sutton, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, San Mateo, CA, 1990. Morgan Kaufmann.

[86] Sutton, R. S. Integrated modeling and control based on reinforcement learning and dynamic programming. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S, editors, *Advances in Neural Information Processing Systems 3*, pages 471–478, San Mateo, CA, 1991. Morgan Kaufmann.

[87] Sutton, R. S. Planning by incremental dynamic programming. In Birnbaum, L. A. and Collins, G. C, editors, *American Association for Artificial Intelligence*, pages 353–357, San Mateo, CA, 1991. Morgan Kaufmann.

[88] Sutton, R. S., Barto, A. G., and Williams, R. J. Reinforcement learning is direct adaptive optimal control. In *Proceedings of the American Control Conference*, pages 2143–2146, Boston, MA, 1991.

[89] Tesauro, G. J. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation.* To appear.

[90] Tesauro, G. J. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4):257–277, May 1992.

[91] Tsitsiklis, J. N. Asynchronous stochastic approximation and Q-learning. Technical Report LIDS-P-2172, Laboratory for Information and Decision Systems, MIT, Cambridge, MA, 1993.

[92] Watkins, C. J. C. H. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, Cambridge, England, 1989.

[93] Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 8(3/4):257–277, May 1992.

[94] Werbos, P. J. Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(1):7–20, 1987.

[95] Werbos, P. J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

[96] Werbos, P. J. Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, 3(2):179–190, 1990.

[97] Werbos, P. J. Approximate dynamic programming for real-time control and neural modeling. In White, D. A. and Sofge, D. A, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pages 493–525. Van Nostrand Reinhold, New York, 1992.

[98] Whitehead, S. D. and Ballard, D. H. Active perception and reinforcement learning. *Neural Computation*, 2:409–419, 1990.

[99] Whitehead, S. D. and Lin, L.-J. Reinforcement learning in non-markov environments. *Artificial Intelligence.* Submitted.

[100] Whittle, P. *Optimization over time: Dynamic programming and stochastic control.* Wiley, New York, 1982.

[101] Williams, R. J. and Baird, L. C. Analysis of some incremental variants of policy iteration: First steps toward understanding Actor-Critic learning systems. Technical Report NU-CCS-93-11, Northeastern University College of Computer Science, 1993.

[102] Williams, R. J. and Baird, L. C. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, Northeastern University College of Computer Science, 1993.

[103] Winston, P. H. *Artificial Intelligence, second edition.* Addison-Wesley Publishing Company, Reading, MA, 1984.

[104] Young, P. *Recursive estimation and time-series analysis.* Springer–Verlag, 1984.